# Contents

Abstract

Preface ................................................................................................................. xlv

   Audience ............................................................................................................. xlv
   Documentation Accessibility ................................................................................ xlv
   Related Documents .............................................................................................. xlv
   Conventions ......................................................................................................... xlvii

News ....................................................................................................................... xlix

   New and Changed Features for Release 12 (11.12.x.0.0) .......................................... xlix
   Other Significant Changes in this Document for Release 12 (11.12.x.0.0) ................. xlix

1 Get Started with Oracle Fusion Applications

   1.1 Overview of Fusion Technologies .................................................................. 1-1

2 Set Up Your Development Environment

   2.1 Introduction to the Development Environment ............................................... 2-1
      2.1.1 Shared Environment ................................................................................ 2-2
      2.1.2 Personal Environment ............................................................................ 2-3
   2.2 Set Up the JDeveloper-based Personal Environment ........................................ 2-4
      2.2.1 Before You Begin .................................................................................... 2-4
      2.2.2 How to Use the Oracle Fusion Domain Wizard ....................................... 2-17
      2.2.3 How to Start Integrated WebLogic Server ............................................. 2-40
   2.3 Improve Shared and Personal Environments Performance ............................... 2-42
      2.3.1 How to Create the OWSM_MDS Schema ............................................... 2-42
      2.3.2 How to Use Alternate Database Schemas ............................................... 2-49
      2.3.3 How to Set Up the Personal Environment for Standalone WebLogic Server ... 2-50
   2.4 Use Deployment Profiles Settings ................................................................... 2-60
      2.4.1 How to Use Service Deployments .......................................................... 2-61
      2.4.2 How to Update the Standard ................................................................. 2-61
   2.5 Configure the Oracle Enterprise Scheduler (ESS) ............................................. 2-62
      2.5.1 How to Provision the Runtime Environment .......................................... 2-62
      2.5.2 How to Create Supporting Database Schema ......................................... 2-63
## 2.6 Test Your Installation

2.7 Use Best Practices for Setting Up the Development Environment

2.7.1 How to Implement Best Practices for JDeveloper

2.7.2 How to Refresh the Oracle ADF Library Dependencies Library

2.7.3 How to Manage OutOfMemory Exceptions (PermGen)

2.7.4 How to Work with Oracle ADF Libraries at Design Time

## 3 Set Up Your JDeveloper Application Workspace and Projects

3.1 Using Technology Scopes

3.2 Provisioning the Application Workspace

3.3 Add Necessary Libraries to Your Data Model Project

3.4 Add the Applications Core Tag Library to Your User Interface Project

3.5 Integrating Oracle Fusion Middleware Extensions for Applications (Applications Core)

3.5.1 What You May Need to Know About Setup UIs in Oracle Fusion Functional Setup Manager

3.5.2 How to Integrate Setup UIs into Functional Setup Manager

3.6 Creating or Updating a Database Connection

3.7 Adding the Search Navigation Tab to the Overview Editor for Oracle Enterprise Crawl and Search Framework (ECSF)

3.7.1 How to Add the Search Navigation Tab to the Overview Editor

3.7.2 What Happens When You Add the Search Navigation Tab to the Overview Editor

3.8 Overriding the Default Resource Bundle

3.9 Implementing Oracle Enterprise Scheduler Service Workspace and Deployment

3.9.1 How to Create the SuperEss Project

3.9.2 How to Build the EAR/MAR Profiles

3.10 Implementing Oracle Application Development Framework UI Workspace and Projects

3.10.1 How to Set Up Your Web Project

3.10.2 How to Create the SuperEss Project in the ADF UI Workspace

3.10.3 How to Deploy Your Web Project

## 4 Get Started with Business Services

4.1 Introduction to Implementing Business Logic

4.1.1 About Entity Objects

4.1.2 About View Objects

4.1.3 About Application Modules

4.2 Understanding Validators

4.3 Understanding List of Values (LOV)

4.4 Understanding Batch Processing

4.5 Understanding Extensibility and Reusability

4.6 Understanding Services

4.7 About Using the Declarative Approach
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.4.1 About Using APIs to Facilitate DML Operations</td>
<td>9-12</td>
</tr>
<tr>
<td>9.4.2 How to Use the Extensions</td>
<td>9-12</td>
</tr>
<tr>
<td>9.4.3 What Happens with PL/SQL Entities at Design Time and Runtime</td>
<td>9-14</td>
</tr>
<tr>
<td>9.5 About Accessing FND Services</td>
<td>9-14</td>
</tr>
<tr>
<td>9.5.1 About Using the Extension</td>
<td>9-14</td>
</tr>
<tr>
<td>9.6 About Using Unique ID</td>
<td>9-15</td>
</tr>
<tr>
<td>9.6.1 About Using the Extension</td>
<td>9-15</td>
</tr>
<tr>
<td>9.6.2 What Happens with Unique ID at Design Time</td>
<td>9-15</td>
</tr>
<tr>
<td>9.6.3 What Happens with Unique ID at Runtime</td>
<td>9-16</td>
</tr>
<tr>
<td>9.7 About Using Data Security</td>
<td>9-16</td>
</tr>
<tr>
<td>9.7.1 How to Use the Extension</td>
<td>9-16</td>
</tr>
<tr>
<td>9.8 About Using Document Sequencing</td>
<td>9-17</td>
</tr>
<tr>
<td>10 Implement Lookups</td>
<td></td>
</tr>
<tr>
<td>10.1 Introduction to Lookups</td>
<td>10-1</td>
</tr>
<tr>
<td>10.1.1 Overview of Lookups</td>
<td>10-1</td>
</tr>
<tr>
<td>10.1.2 About Standard, Set-Enabled, and Common Lookup Views</td>
<td>10-3</td>
</tr>
<tr>
<td>10.1.3 About Lookup Customization Levels</td>
<td>10-5</td>
</tr>
<tr>
<td>10.2 About Preparing Entities and Views for Lookups</td>
<td>10-6</td>
</tr>
<tr>
<td>10.2.1 How to Prepare Custom Lookup Views</td>
<td>10-6</td>
</tr>
<tr>
<td>10.3 About Referencing Lookups</td>
<td>10-8</td>
</tr>
<tr>
<td>10.3.1 How to Reference Lookups</td>
<td>10-8</td>
</tr>
<tr>
<td>10.4 About Defining Validators for Lookups</td>
<td>10-8</td>
</tr>
<tr>
<td>10.4.1 How to Define a List Validator</td>
<td>10-9</td>
</tr>
<tr>
<td>10.4.2 About Defining a Key Exists Validator</td>
<td>10-11</td>
</tr>
<tr>
<td>10.5 Annotate Lookup Code Reference Attributes for Set-Enabled Lookups</td>
<td>10-14</td>
</tr>
<tr>
<td>10.6 About Integrating Lookups Task Flows into Oracle Fusion Functional Setup Manager</td>
<td>10-15</td>
</tr>
<tr>
<td>11 Set Up Document Sequences</td>
<td></td>
</tr>
<tr>
<td>11.1 Introduction to Document Sequences</td>
<td>11-1</td>
</tr>
<tr>
<td>11.2 About Defining Document Sequence Categories</td>
<td>11-2</td>
</tr>
<tr>
<td>11.3 About Assigning a Document Sequence</td>
<td>11-2</td>
</tr>
<tr>
<td>11.4 About Striping Document Sequence Assignments</td>
<td>11-3</td>
</tr>
<tr>
<td>11.5 About Defining a Document Sequence Audit Table</td>
<td>11-3</td>
</tr>
<tr>
<td>11.6 About Enabling Document Sequences in ADF Business Components</td>
<td>11-4</td>
</tr>
<tr>
<td>11.6.1 About Using the Document-Sequence Extension</td>
<td>11-4</td>
</tr>
<tr>
<td>11.7 About Managing PL/SQL APIs</td>
<td>11-7</td>
</tr>
<tr>
<td>11.8 About Integrating Document Sequence Task Flows into Oracle Fusion Functional Setup Manager</td>
<td>11-9</td>
</tr>
<tr>
<td>12 Implement Audit Reports</td>
<td></td>
</tr>
<tr>
<td>12.1 Introduction to Audit Reports</td>
<td>12-1</td>
</tr>
<tr>
<td>12.2 Implement Audit Reports in the Oracle Fusion Applications Framework</td>
<td>12-1</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>12.2.1</td>
<td>How to Set Metadata</td>
</tr>
<tr>
<td>12.2.2</td>
<td>How to Set Up Auditing</td>
</tr>
<tr>
<td>12.3</td>
<td>Using Audit Reports</td>
</tr>
<tr>
<td>12.3.1</td>
<td>How to Search in Audit Reports</td>
</tr>
<tr>
<td>13</td>
<td>Get Started with Your Web Interface</td>
</tr>
<tr>
<td>13.1</td>
<td>Introduction to Developing a Web Application</td>
</tr>
<tr>
<td>13.2</td>
<td>Oracle Fusion Guidelines, Patterns, and Standards</td>
</tr>
<tr>
<td>13.3</td>
<td>Basic Building Blocks</td>
</tr>
<tr>
<td>13.4</td>
<td>Introduction to the UI Shell</td>
</tr>
<tr>
<td>13.5</td>
<td>Applications UI Patterns and Features</td>
</tr>
<tr>
<td>14</td>
<td>Implement the UI Shell</td>
</tr>
<tr>
<td>14.1</td>
<td>Introduction to Implementing the UI Shell</td>
</tr>
<tr>
<td>14.1.1</td>
<td>Standard Related to the UI Shell</td>
</tr>
<tr>
<td>14.1.2</td>
<td>UI Shell Description</td>
</tr>
<tr>
<td>14.2</td>
<td>Populating a UI Shell</td>
</tr>
<tr>
<td>14.2.1</td>
<td>How to Create a JSF Page</td>
</tr>
<tr>
<td>14.2.2</td>
<td>How to Add Default Main Area Task Flows to a Page</td>
</tr>
<tr>
<td>14.2.3</td>
<td>How to Add Dynamic Main Area and Regional Area Task Flows to a Page</td>
</tr>
<tr>
<td>14.2.4</td>
<td>How to Pass Parameters into Task Flows from Tasks List</td>
</tr>
<tr>
<td>14.2.5</td>
<td>How to Open Data Files from a Tasks List Link</td>
</tr>
<tr>
<td>14.3</td>
<td>Implementing Application Menu Security</td>
</tr>
<tr>
<td>14.4</td>
<td>Controlling the State of Main and Regional Area Task Flows</td>
</tr>
<tr>
<td>14.4.1</td>
<td>How to Control Main Area Task Flows</td>
</tr>
<tr>
<td>14.4.2</td>
<td>How to Control Regional Area Task Flows</td>
</tr>
<tr>
<td>14.4.3</td>
<td>How to Control the State of the Contextual Area Splitter</td>
</tr>
<tr>
<td>14.4.4</td>
<td>Sizing Regional Area Panels</td>
</tr>
<tr>
<td>14.5</td>
<td>Working with the Global Menu Model</td>
</tr>
<tr>
<td>14.5.1</td>
<td>How to Implement a Global Menu</td>
</tr>
<tr>
<td>14.5.2</td>
<td>How to Set Up Global Menu Security</td>
</tr>
<tr>
<td>14.5.3</td>
<td>How to Create the Navigator</td>
</tr>
<tr>
<td>14.6</td>
<td>Using the Settings and Actions Menu</td>
</tr>
<tr>
<td>14.6.1</td>
<td>How to Use the Personalization Options</td>
</tr>
<tr>
<td>14.6.2</td>
<td>How to Implement End User Preferences</td>
</tr>
<tr>
<td>14.6.3</td>
<td>How to Use the Administration Options</td>
</tr>
<tr>
<td>14.6.4</td>
<td>How to Use the Troubleshooting Options</td>
</tr>
<tr>
<td>14.6.5</td>
<td>How to Set Up a Proxy User (Session Impersonation)</td>
</tr>
<tr>
<td>15</td>
<td>Implement Search Functions in the UI Shell</td>
</tr>
<tr>
<td>15.1</td>
<td>Adding Attachments Support to Global Search</td>
</tr>
<tr>
<td>15.1.1</td>
<td>How ECSF Crawl Operates</td>
</tr>
<tr>
<td>15.1.2</td>
<td>How the Global Search Results UI Shows Existing Attachments</td>
</tr>
</tbody>
</table>
15.1.3 How Crawling Is Done................................................................. 15-5
15.1.4 How Attachments Are Shown.................................................. 15-10
15.1.5 How Attachments Are Opened............................................... 15-10
15.1.6 How to Implement Attachments Search................................. 15-11
15.2 Implementing Recent Items .......................................................... 15-13
  15.2.1 How to Choose Labels for Task Flows................................. 15-13
  15.2.2 How to Call Sub Flows.......................................................... 15-13
  15.2.3 How to Enable a Sub Flow to Be Bookmarked in Recent Items...... 15-15
  15.2.4 How to Use Additional Capabilities of the Recent Items openSubTask API........ 15-21
  15.2.5 How to Implement Data Security for Recent Items and Favorites ............... 15-22
  15.2.6 Known Issues......................................................................... 15-22
15.3 Implementing the Watchlist.......................................................... 15-22
  15.3.1 Watchlist Data Model Effects .................................................. 15-24
  15.3.2 Watchlist Physical Data Model Entities..................................... 15-24
  15.3.3 Supported Watchlist Items...................................................... 15-28
  15.3.4 How to Use the Watchlist....................................................... 15-31
15.4 Implementing Group Spaces.......................................................... 15-42
  15.4.1 How to Implement Group Spaces............................................ 15-43
  15.4.2 Overview of Group Spaces Functionality............................... 15-43
  15.4.3 How to Pass a Chromeless Template...................................... 15-44
15.5 Implementing Activity Streams and Business Events...................... 15-44
  15.5.1 Introduction to WebCenter Portal Activities........................... 15-44
  15.5.2 How to Publish Business Events to Activities.......................... 15-45
  15.5.3 How to Publish Activities Using a Programmatic API.................. 15-45
  15.5.4 How to Implement Activity Streams........................................ 15-49
  15.5.5 How to Define Activities........................................................ 15-52
  15.5.6 How to Implement Comments and Likes.................................. 15-57
  15.5.7 How to Implement Follow for an Object................................... 15-57
  15.5.8 How to Render Contextual Actions in Activity Streams............. 15-59
15.6 Implementing the Oracle Fusion Applications Search Results UI........ 15-60
  15.6.1 How to Disable Oracle Fusion Applications Search.................. 15-60
  15.6.2 How to Use Basic Search...................................................... 15-61
  15.6.3 How to Control the Look and Feel of Search........................... 15-69
  15.6.4 Introduction to the Crawled Objects Project............................ 15-82
  15.6.5 How to Implement Tags in Oracle Fusion Applications Search......... 15-83
  15.6.6 How to Use the Actionable Results API with Oracle Fusion Applications Search 15-87
  15.6.7 How to Integrate Non-Applications Data into Oracle Fusion Applications Search 15-99
15.7 Implementing Auto-suggest......................................................... 15-100
  15.7.1 How to Implement fnd:searchSuggestBehavior......................... 15-107
  15.7.2 How to Implement a Custom Suggestion Group......................... 15-110
  15.7.3 How to Implement fnd:searchSuggestBehavior in Global Search........... 15-112
  15.7.4 How to Implement Dynamic UI Support for searchSuggestBehavior ....... 15-114
15.8 Attachments in Global Search....................................................... 15-125
15.8.1 Crawl Functionality ...................................................................................................... 15-125
15.8.2 Show Attachments ........................................................................................................ 15-130
15.8.3 Open Attachments ........................................................................................................ 15-130
15.8.4 Implement the Attachment Plugin ............................................................................. 15-131
15.9 Support Global Search Alternate Words for the Cloud ...................................................... 15-132
15.9.1 Look Up the SES Endpoint (ECSF APIs) .................................................................... 15-134
15.9.2 SES Web Service Calls ................................................................................................. 15-135
15.9.3 Alternate Words Error Messages ................................................................................ 15-137
15.10 Implement List Search .................................................................................................... 15-137
15.10.1 How to Implement List Search at Design Time ....................................................... 15-144

16 Implement Additional Functions in the UI Shell

16.1 Introducing the Navigate API .............................................................................................. 16-1
16.1.1 How to Use the Navigate API Data Control Method .................................................. 16-2
16.1.2 How to Implement Navigation Across Web Applications ........................................... 16-8
16.2 Warning of Pending Changes in the UI Shell .................................................................... 16-8
16.2.1 How to Implement Warning of Pending Changes ...................................................... 16-9
16.2.2 How to Suppress Warning of Pending Changes ....................................................... 16-10
16.3 Implementing the Oracle Fusion Home Page UI ................................................................. 16-10
16.3.1 Supported Behavior .................................................................................................... 16-11
16.3.2 How to Create a Home Page ....................................................................................... 16-11
16.3.3 Getting the URL ........................................................................................................... 16-12
16.4 Using the Single Object Context Workarea ....................................................................... 16-12
16.4.1 Implementation Notes ................................................................................................ 16-13
16.5 Implementing the Third-Party Component Area ............................................................... 16-15
16.5.1 How to Implement the ThirdPartyComponentArea Facet Developer ............................ 16-15
16.6 Developing an Activity Guide Client Application with the UI Shell ................................. 16-16
16.6.1 How to Develop an Activity Guide Client Application Using Oracle UI Shell ............. 16-16
16.6.2 How to add a task flow to the Oracle UI Shell client application ............................... 16-20
16.6.3 How to Secure the Activity Guide Oracle UI Shell Client Application ........................ 16-23
16.7 Implementing Touch Device Support in the UI Shell .......................................................... 16-24
16.8 Implementing the Translation Editor ................................................................................. 16-25
16.8.1 How to Implement the Translation Editor .................................................................. 16-27
16.8.2 How to Add the Translation Editor to Existing Application Tables During Design Time ................................................................. 16-31
16.8.3 How to Add the Translation Editor to New Application Tables During Design Time ................................. 16-33
16.8.4 Handle Multi-Language Data Entry ............................................................................ 16-33
16.9 About the Create Page Wizard .......................................................................................... 16-51
16.9.1 How to Complete the Create Page Wizard ................................................................. 16-51
16.10 Troubleshooting UI Shell Issues ..................................................................................... 16-53
16.10.1 ApplSession Is Not Created ...................................................................................... 16-53
16.10.2 Navigator Shows a Little White Box .......................................................................... 16-54
17 Implement UIs in JDeveloper with Application Tables, Trees and Tree Tables

17.1 Implementing Applications Tables ................................................................. 17-1
  17.1.1 Understanding Applications Tables Facets and Properties ....................... 17-2
  17.1.2 How to Create an Applications Table ....................................................... 17-11
  17.1.3 Introduction to Selected Elements in the Table Property Inspector ............... 17-23
  17.1.4 How to Modify Applications Table Components and Properties ............... 17-26

17.2 Implementing the Applications Tree ............................................................... 17-28
  17.2.1 How to Add an Applications Tree to Your Page .......................................... 17-28

17.3 Implementing Applications Tree Tables ......................................................... 17-41
  17.3.1 How to Add an Applications Tree Table ..................................................... 17-50

17.4 Using the Custom Wizard with Applications Popups ......................................... 17-56
  17.4.1 Creating a Popup .......................................................................................... 17-57
  17.4.2 How to Modify Popup Components and Properties ...................................... 17-63

18 Implement Applications Panels, Master-Detail, Hover, and Dialog Details

18.1 Implementing Applications Panels .................................................................. 18-1
  18.1.1 Overview of Applications Panel Components .............................................. 18-1
  18.1.2 How to Create an Applications Panel ......................................................... 18-11
  18.1.3 How to Modify Applications Panels Components and Properties ............... 18-19

18.2 Implementing the Simple Panel ....................................................................... 18-21
  18.2.1 How to Implement the Simple Panel ............................................................. 18-29
  18.2.2 How to Add Simple Panel at Design Time ................................................... 18-30

18.3 Implementing Applications Master-Detail ....................................................... 18-44
  18.3.1 Component Structure and Functions ............................................................ 18-44
  18.3.2 Introduction to Master-Detail Components ............................................... 18-45
  18.3.3 How to Create a Master-Detail ................................................................. 18-46
  18.3.4 Master-Detail Guidelines for Creating New Records ................................... 18-55
  18.3.5 How to Modify Master-Detail Components and Properties ......................... 18-56

18.4 Implementing Hover ....................................................................................... 18-56

18.5 Implementing Applications Dialog Details ...................................................... 18-59
  18.5.1 How to Add Applications Dialog Details to Your Page ............................... 18-59

19 Implement Attachments

19.1 Introduction to Attachment Components ....................................................... 19-1
  19.1.1 About the Attachment Views Component ............................................... 19-2
  19.1.2 About the Attachment Link Component ............................................... 19-8
  19.1.3 About the Attachment Component ........................................................... 19-10
  19.1.4 About the Attachments Carousel ............................................................ 19-15

19.2 About Creating Attachments ......................................................................... 19-16
  19.2.1 Performing Prerequisite Tasks ................................................................. 19-16
  19.2.2 Understanding Content Repository Prerequisites ..................................... 19-19
19.2.3 About Setting Up Your Model Project for Attachments................................. 19-20
19.2.4 How to Create Attachment View Links............................................................... 19-21
19.2.5 What Happens When You Create an Attachment View Link............................. 19-39
19.2.6 About Deleting the Business Object ................................................................. 19-40
19.2.7 About Assigning Categories to the Attachment Entity........................................ 19-41
19.2.8 How to Create an Attachments Field, Attachments Table, Attachments Carousel,
or Attachment Views.............................................................................................................. 19-41
19.2.9 What Happens When You Implement Attachments ............................................ 19-42
19.2.10 How to Create an Attachments Column in an Applications Table .................... 19-43
19.2.11 How to Create an Attachments Column in an Existing Applications Table........ 19-44
19.2.12 About Setting Up Required Properties ............................................................. 19-44
19.2.13 What Happens at Runtime................................................................................. 19-45
19.2.14 How to Update Page Bindings for the Attachment Views Component................ 19-45
19.3 About Displaying Attachments for Multiple Entities in the Same Table................ 19-46
19.3.1 About Configuring the Attachments Component to Display Attachments for
Multiple Entities....................................................................................................................... 19-48
19.4 About Configuring Attachments Component Properties........................................ 19-50
19.4.1 About Using FUSE ................................................................................................. 19-50
19.5 About Working with Attachments Programmatically............................................ 19-51
19.5.1 About Creating New Attachment Types ............................................................... 19-52
19.5.2 About Retrieving Attachments............................................................................. 19-55
19.5.3 About Using Attachment Utilities........................................................................ 19-56
19.5.4 About Managing Attachments HTTP Sessions.................................................... 19-57
19.5.5 About Creating Third-Party Attachments............................................................ 19-57
19.5.6 About Implementing Attachments with PL/SQL.................................................. 19-59
19.6 About Setting Up Miscellaneous Attachments Features......................................... 19-62
19.6.1 About Custom Actions for the Attachments Component..................................... 19-62
19.6.2 About Custom Actions for the Attachment Views and Attachment Link
Component.............................................................................................................................. 19-63
19.6.3 About Approvals..................................................................................................... 19-64
19.6.4 About Redlining.................................................................................................... 19-65
19.7 About Integrating Attachments Task Flows into Oracle Fusion Functional Setup
Manager................................................................................................................................. 19-67
19.8 About Securing Attachments................................................................................ 19-68
19.8.1 About Entity Instance-Level Security................................................................. 19-68
19.8.2 About Attachment Category Data Security.......................................................... 19-69
19.8.3 About File Sharing ............................................................................................... 19-70
19.8.4 About File Format Filtering.................................................................................. 19-70
19.8.5 About Attachments SaaS..................................................................................... 19-71
19.9 Understanding How to Use Attachments (Runtime)............................................... 19-71
19.9.1 About Using Attachments File-Level Security..................................................... 19-71
19.9.2 About Updating Attachments............................................................................... 19-72
19.9.3 How to Check Out and Check In File Attachments.............................................. 19-75
Organize Hierarchical Data with Tree Structures

20.1 Introduction to Trees
  20.1.1 Understanding Tree Structures, Trees, and Tree Versions
  20.2 About Configuring the Trees Application Launch Page
    20.2.1 How to Create the Launch Page
  20.3 Working with Tree Structures
    20.3.1 About Managing Tree Structure Data Sources
    20.3.2 About Specifying Data Source Parameters
    20.3.3 About Using the Search Field
    20.3.4 About Using the Search Field
    20.3.5 How to Create a Tree Structure
    20.3.6 How to Duplicate a Tree Structure
    20.3.7 How to Edit a Tree Structure
    20.3.8 How to Delete a Tree Structure
  20.4 Working with Trees
    20.4.1 How to Search for a Tree
    20.4.2 How to Create a Tree
    20.4.3 How to Duplicate a Tree
    20.4.4 How to Edit a Tree
    20.4.5 How to Delete a Tree
  20.5 Working with Tree Versions
    20.5.1 How to Create a Tree Version
    20.5.2 About Adding Tree Nodes to a Tree Version
    20.5.3 How to Create a Record for a Data Source
    20.5.4 How to Duplicate a Tree Version
    20.5.5 How to Edit a Tree Version
    20.5.6 About Performing CRUD Operations on Tree Nodes Using APIs
    20.5.7 About Performing Sub-tree Node Operations Using PL/SQL APIs
    20.5.8 How to Set Tree Version Status
    20.5.9 About Auditing Trees and Tree Versions
    20.5.10 About Flattening Rows and Columns
  20.6 Manage Labels in the Generic Label Data Source
    20.6.1 How to Search for a Label
    20.6.2 How to Create a Label
    20.6.3 How to Edit a Label
    20.6.4 How to Delete a Label
  20.7 About Using the Applications Hierarchy Component to Develop Applications
    20.7.1 How to Create a Tree Application
    20.7.2 How to Create a Tree Table Application
    20.7.3 How to Sort Hierarchies
24.6 Create and Configuring Extensible Flexfield View Objects ......................................................... 24-20
24.6.1 How to Create and Configure Extensible Flexfield View Objects ............................................. 24-20
24.7 Defining and Registering Extensible Flexfields ............................................................................. 24-23
24.7.1 How to Register Extensible Flexfields ....................................................................................... 24-23
24.8 Creating Extensible Flexfield Business Components ................................................................. 24-28
24.8.1 How to Create Business Components For an Extensible Flexfield’s Usage ......................... 24-30
24.9 Employ an Extensible Flexfield on a User Interface Page ............................................................ 24-33
24.9.1 How to Expose an Extensible Flexfield Usage as a Single Task Flow for Multiple Contexts ........................................................................................................................................ 24-34
24.9.2 How to Expose the Complete Set of an Extensible Flexfield’s Usages, Logical Pages, and Associated Contexts .................................................................................................................... 24-36
24.9.3 How to Expose One Extensible Flexfield Context .................................................................... 24-39
24.9.4 How to Expose Extensible Flexfield Attributes in a Table ...................................................... 24-41
24.10 Incorporating Extensible Flexfields into a Search Form .............................................................. 24-43
24.10.1 How to Incorporate an Extensible Flexfield into a Search Form ............................................ 24-43
24.11 Preparing Extensible Flexfield Business Components for Oracle Business Intelligence ............ 24-44
24.11.1 How to Enable an Extensible Flexfield for Oracle Business Intelligence .............................. 24-45
24.11.2 How to Flatten the Extensible Flexfield Model for a Business Intelligence-Enabled Extensible Flexfield ................................................................................................................. 24-45
24.12 Integrating Extensible Flexfields with Business Intelligence Publisher ........................................ 24-47
24.13 Loading Seed Data ....................................................................................................................... 24-48
24.14 Customize the Extensible Flexfield Modelers ............................................................................. 24-48
24.14.1 How to Customize the Runtime Business Component Modeler for Extensible Flexfields .......... 24-48
24.14.2 How to Customize the Runtime User Interface Modeler for Extensible Flexfields ................. 24-50
24.15 Testing Flexfields .......................................................................................................................... 24-54
24.16 Making Extensible Flexfields Available to Web Services ............................................................ 24-54
24.16.1 How to Make an Extensible Flexfield Available to Web Services ........................................ 24-54
24.17 Access Information About Extensible Flexfield Business Components .................................... 24-56
24.17.1 How to Access Information About Extensible Flexfield Business Components ................ 24-56
24.18 Use the Advanced Features of Extensible Flexfields .................................................................... 24-58
24.18.1 How to Support Interface Loading of Extensible Flexfield Data .............................................. 24-58
24.18.2 How to Support Parallel Sets of Extensible Flexfield Artifacts for Different Uses .................. 24-60

25 Use Key Flexfields

25.1 Introduction to Key Flexfields ....................................................................................................... 25-1
25.1.1 Benefits of Key Flexfields ........................................................................................................ 25-1
25.1.2 How Key Flexfields Are Modeled in Oracle Application Development Framework .......... 25-2
25.1.3 Secondary Usage Feature ........................................................................................................ 25-2
25.1.4 Participant Roles ...................................................................................................................... 25-3
25.1.5 Complete the Key Flexfield Development Process ................................................................... 25-3
25.2 Complete the Producer Tasks for Key Flexfields

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.2.1 How to Develop Key Flexfields</td>
<td>25-8</td>
</tr>
<tr>
<td>25.2.2 How to Implement Key Flexfield Segment Labels</td>
<td>25-13</td>
</tr>
<tr>
<td>25.2.3 How to Implement Cross-Validation Rules and Custom Validation</td>
<td>25-15</td>
</tr>
<tr>
<td>25.2.4 How to Create Key Flexfield Business Components</td>
<td>25-19</td>
</tr>
<tr>
<td>25.2.5 How to Share Key Flexfield Business Components</td>
<td>25-19</td>
</tr>
<tr>
<td>25.2.6 How to Build a Key Flexfield Maintenance User Interface</td>
<td>25-37</td>
</tr>
<tr>
<td>25.2.7 What Happens at Runtime: Creating New Combinations</td>
<td>25-38</td>
</tr>
</tbody>
</table>

25.3 Complete the Consumer Tasks for Key Flexfields in Reference Mode

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.3.1 How to Create Key Flexfield View Links</td>
<td>25-39</td>
</tr>
<tr>
<td>25.3.2 How to Nest an Instance of the Key Flexfield Application Module in the Product Application Module</td>
<td>25-41</td>
</tr>
<tr>
<td>25.3.3 How to Add an Instance of a Key Flexfield View Object to the Product Application Module</td>
<td>25-42</td>
</tr>
</tbody>
</table>

25.4 Employ Key Flexfield UI Components on a Page

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.4.1 How to Employ a Key Flexfield Component on a Page</td>
<td>25-43</td>
</tr>
<tr>
<td>25.4.2 How to Incorporate Key Flexfields into a Query Search Form</td>
<td>25-51</td>
</tr>
<tr>
<td>25.4.3 How to Configure Key Flexfield UI Components</td>
<td>25-57</td>
</tr>
</tbody>
</table>

26 Use the Advanced Features of Key Flexfields

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>26.1 Introduction to Advanced Features of Key Flexfields</td>
<td>26-1</td>
</tr>
<tr>
<td>26.2 Use Key Flexfield Advanced Features in Reference Mode</td>
<td>26-1</td>
</tr>
<tr>
<td>26.2.1 How to Define Code Combination Constraints</td>
<td>26-1</td>
</tr>
<tr>
<td>26.2.2 How to Access Segment Labels Using the Java API</td>
<td>26-9</td>
</tr>
<tr>
<td>26.2.3 How to Prepare Key Flexfield Business Components for Oracle Business Intelligence</td>
<td>26-10</td>
</tr>
<tr>
<td>26.2.4 How to Publish Key Flexfield Application Modules as Web Services</td>
<td>26-14</td>
</tr>
<tr>
<td>26.2.5 How to Access Key Flexfields from an ADF Desktop Integration Excel Workbook</td>
<td>26-26</td>
</tr>
</tbody>
</table>

26.3 Complete the Development Tasks for Key Flexfields in Secondary Mode

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>26.3.1 How to Register a Key Flexfield All-Segment Secondary Usage</td>
<td>26-34</td>
</tr>
<tr>
<td>26.3.2 How to Register a Key Flexfield Single-Segment Secondary Usage</td>
<td>26-35</td>
</tr>
<tr>
<td>26.3.3 How to Create Key Flexfield Business Components for Secondary Usage</td>
<td>26-35</td>
</tr>
<tr>
<td>26.3.4 How to Create Key Flexfield View Links for a Secondary Usage</td>
<td>26-40</td>
</tr>
</tbody>
</table>

26.4 Work with Code-Combination Filters for Key Flexfields

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>26.4.1 How to Use Standard Combination Filters</td>
<td>26-42</td>
</tr>
<tr>
<td>26.4.2 How to Use Code-Combination Filters for Oracle BI Publisher Reports</td>
<td>26-44</td>
</tr>
<tr>
<td>26.4.3 How to Use Cross-Validation Filters</td>
<td>26-44</td>
</tr>
<tr>
<td>26.4.4 How to Prepare the Database for Standard Code-Combination Filters</td>
<td>26-45</td>
</tr>
<tr>
<td>26.4.5 How to Add Code-Combination Filters to Your Application</td>
<td>26-46</td>
</tr>
<tr>
<td>26.4.6 How to Employ Code-Combination Filters on an Application Page</td>
<td>26-52</td>
</tr>
<tr>
<td>26.4.7 How to Create Code-Combination Filter Definitions for Testing</td>
<td>26-56</td>
</tr>
<tr>
<td>26.4.8 How to Apply Code-Combination Filters Using the PL/SQL Filter APIs</td>
<td>26-58</td>
</tr>
</tbody>
</table>
26.4.9   How to Use an API to Apply XML Filters ................................................................. 26-64
26.4.10  How to Remove Code-Combination Filters from Your Application............... 26-66
26.4.11  How to Remove Filters from the Filter Repository .................................................. 26-66

27   Test and Deploy Flexfields

27.1   Introduction to Testing and Deploying Flexfields.................................................... 27-1
27.2   Deploy and Run the Flexfield Configuration User Interface .................................... 27-2
  27.2.1  How to Deploy and Run the ApplCore Setup Application to Integrated WebLogic
          Server ......................................................................................................................... 27-2
27.3   Configure Flexfields ................................................................................................. 27-4
  27.3.1  How to Configure Flexfields for Testing ............................................................. 27-4
27.4   Test Flexfields ........................................................................................................... 27-5
  27.4.1  Test Flexfields from Oracle JDeveloper .............................................................. 27-5
  27.4.2  Test Flexfields from Integrated WebLogic Server ............................................... 27-7
27.5   Deploy Flexfields in a Standalone WebLogic Server Environment ....................... 27-10
  27.5.1  How to Package a Flexfield Application for Deployment .................................... 27-10
  27.5.2  How to Deploy a Flexfield Application ................................................................. 27-11
27.6   Use the WLST Flexfield Commands to Deploy Flexfields...................................... 27-16
  27.6.1  How to Prepare Your Environment to Use the WLST Flexfield Commands ...... 27-17
  27.6.2  How to Prepare Your Environment to Use the deployFlexForApp Command ... 27-17
27.7   Regenerate Flexfield Business Components Programmatically ............................ 27-18
27.8   Integrate Flexfield Task Flows into Oracle Fusion Functional Setup Manager.......... 27-19
27.9   Export Flexfield Business Components .................................................................. 27-22

28   Get Started with Oracle Enterprise Crawl and Search Framework

28.1   Introduction to Using Oracle Enterprise Crawl and Search Framework ............... 28-1
  28.1.1  Understanding the ECSF Architecture ................................................................. 28-1
28.2   Setting Up and Running ECSF Command Line Administration Utility ............... 28-5
  28.2.1  Set Up the ECSF Command Line Administration Utility .................................. 28-5
  28.2.2  Run the ECSF Command Line Administration Utility ....................................... 28-6
  28.2.3  About Making Searchable Objects Accessible to the ECSF Command Line
          Administration Utility .................................................................................................. 28-6
  28.2.4  About Setting the Class Path ............................................................................... 28-7
  28.2.5  About Setting the Connection Information ......................................................... 28-9
  28.2.6  About Manually Connecting to the Oracle Fusion Applications Database ....... 28-11
  28.2.7  How to Provide the Path of the JPS Config File .................................................. 28-12
  28.2.8  About Configuring the Log Settings ................................................................... 28-13
  28.2.9  About Automating the ECSF Command Line Administration Utility ............ 28-13
28.3   Set Up Oracle Enterprise Manager and Discovering ECSF ................................. 28-14
  28.3.1  How to Register the ECSF Runtime MBean to the Integrated WebLogic Server .. 28-14
  28.3.2  About Installing Oracle Enterprise Manager ..................................................... 28-16
  28.3.3  How to Discover ECSF in Oracle Enterprise Manager ...................................... 28-16
  28.3.4  About Adding Users to the Administrators Group .............................................. 28-17
31.2 About Validating the Search Metadata .......................................................... 31-1
  31.2.1 How to Validate Search Metadata .......................................................... 31-2
31.3 Testing Searchable Objects Through a Web Browser ........................................ 31-2
  31.3.1 Perform Prerequisite Steps....................................................................... 31-2
  31.3.2 How to Run the ECSF Feed Servlet ......................................................... 31-3
  31.3.3 How to Test the Config Feed ...................................................................... 31-4
  31.3.4 How to Test the Control Feed ..................................................................... 31-5
  31.3.5 How to Test the Data Feed ......................................................................... 31-6
  31.3.6 How to Reset the State of the Feeds.......................................................... 31-7

32 Deploy and Crawl Searchable Objects
  32.1 Introduction to Deploying and Crawling Searchable Objects ......................... 32-1
  32.2 Deploy Searchable Objects and Dependencies................................................ 32-1
    32.2.1 About Deploying the ECSF Shared Library to Oracle WebLogic Server ...... 32-1
    32.2.2 How to Create an Application................................................................. 32-3
    32.2.3 How to Change the Application Name and Context Root of the View-Controller Project ................................................................. 32-4
    32.2.4 How to Modify the Run Configuration of the View-Controller Project ..... 32-4
    32.2.5 About Adding the ECSF Runtime Server Library and Required Java Archive Files to the Model and View-Controller Projects ................................................. 32-7
    32.2.6 About Deploying the ECSF Application.................................................. 32-8
  32.3 About Crawling Searchable Objects ............................................................... 32-8
    32.3.1 How to Verify the Crawl........................................................................... 32-8

33 Advanced Topics for ECSF
  33.1 Introduction to Advanced Topics for ECSF...................................................... 33-1
  33.2 About Enabling Search on Fusion File Attachments....................................... 33-1
    33.2.1 How to Make File Attachments Crawlable ............................................. 33-2
  33.3 About Enabling Search on WebCenter Tags.................................................... 33-2
    33.3.1 How to Enable Search on WebCenter Portal Tags .................................. 33-5
    33.3.2 How to Customize Search on WebCenter Portal Tags ......................... 33-6
    33.3.3 How to Add Tags to Indexable Documents ............................................. 33-6
    33.3.4 About Adding Tags for Querying.............................................................. 33-7
    33.3.5 About Modifying Tags in Indexable Documents..................................... 33-8
    33.3.6 About Registering Change Listeners......................................................... 33-8
  33.4 About Enabling Search on Tree-Structure-based Source Systems .................... 33-9
    33.4.1 How to Enable Search on Tree-Structure-Based Source Systems .......... 33-11
    33.4.2 How to Crawl Tree Structures................................................................. 33-11
    33.4.3 About Integrating Search Functionality for Tree Structures.................... 33-21
  33.5 About Managing Recent Searches................................................................... 33-27
    33.5.1 About Using the RecentSearchManager API ......................................... 33-28
    33.5.2 Understanding How Recent Searches Are Processed............................. 33-29
  33.6 Set Up Federated Search.................................................................................. 33-31
33.6.1 How to Create the SearchDB Connection on Oracle WebLogic Server Instance... 33-32
33.6.2 How to Update the Application Deployment Profile with the Target Directory for
Searchable Objects .................................................................................................................. 33-32
33.6.3 About Updating the Application to Reference the ECSF Service Shared Library. 33-33
33.6.4 How to Add the ECSF Runtime Library ................................................................. 33-33
33.6.5 About Setting the System Parameter for Web Service............................................ 33-34
33.6.6 About Packaging and Deploying the Search Application........................................ 33-34
33.6.7 About Updating the Search Application with New Searchable Objects or
Dependences .......................................................................................................................... 33-35
33.6.8 How to Set Up the ECSF Client Application for Federation................................. 33-35
33.6.9 About Setting the SearchContext Scope to GLOBAL.............................................. 33-38
33.6.10 About Integrating Federation Across Oracle Fusion Applications Product
Families ................................................................................................................................. 33-39
33.7 About Federating Oracle SES Instances ...................................................................... 33-39
33.8 About Raising Change Events Synchronously ............................................................ 33-41
33.9 About Using the External ECSF Web Service for Integration..................................... 33-41
33.9.1 About Web Service Methods .................................................................................... 33-41
33.9.2 About ECSF Web Service WSDL and XSD ............................................................... 33-42
33.9.3 About Web Service Request XSDs and XMLs.......................................................... 33-48
33.9.4 Understanding Web Service Response XSDs .......................................................... 33-55
33.9.5 About Invoking the ECSF Web Service................................................................. 33-65
33.10 About Implementing Coherence Cache for Search ..................................................... 33-70
33.10.1 About Implement Cache Coherence for Search....................................................... 33-71
33.11 About Localizing ECSF Artifacts.................................................................................. 33-71
33.11.1 How to Translate Strings in Groovy Expressions...................................................... 33-71
33.11.2 About Localizing facet Display Names .................................................................... 33-74
33.11.3 About Localizing Crawl Management Display Names.......................................... 33-78
33.11.4 About Localizing Crawlable Dynamic Content ...................................................... 33-78
33.11.5 About Localizing Crawlable Template Content ...................................................... 33-78
33.11.6 About Determining Locale ..................................................................................... 33-80
33.12 About Using ECSF Diagnostics .................................................................................. 33-80
33.12.1 Query Tests ............................................................................................................... 33-81
33.12.2 About Crawl Tests .................................................................................................... 33-83
33.12.3 About Environment and Configuration Information Tests .................................... 33-84
33.12.4 About Security Tests ............................................................................................... 33-85
33.13 About Troubleshooting ECSF .................................................................................... 33-87
33.13.1 Understand Common Problems and Solutions ....................................................... 33-87
33.13.2 About Diagnosing ECSF Problems ........................................................................ 33-93

34 Initiate a SOA Composite from an Oracle ADF Web Application

34.1 Introduction to the Recommended Design Pattern..................................................... 34-1
34.2 Other Approaches ....................................................................................................... 34-3
34.3 Example ....................................................................................................................... 34-3
36.6 Verifying the Deployment
36.7 Troubleshooting the Use Case
36.8 What You May Need to Know About Orchestrating ADF Business Components Services

37 Manipulate Back-End Data from a SOA Composite
37.1 Introduction to the Recommended Design Pattern
37.2 Example
37.3 How to Manipulate Data from a BPEL Process Service Component
37.4 Securing the Design Pattern
37.5 Verifying the Deployment
37.6 Troubleshooting the Use Case
37.7 What You May Need to Know About Manipulating Back-end Data from a SOA Composite
    37.7.1 When Entity Variables Flush Changes Back to ADF Business Components
    37.7.2 Support for XPath Operations
    37.7.3 Invoking an ADF Business Components Service and Entity Variables in the Same BPEL Process Service Component

38 Access a PL/SQL Service from a SOA Composite
38.1 Introduction to the Recommended Design Pattern
38.2 Other Approaches
38.3 Example
38.4 How to Invoke a PL/SQL Stored Procedure from a SOA Composite Application
38.5 Securing the Design Pattern
38.6 Verifying the Deployment

39 Invoke Custom Java Code from a SOA Composite
39.1 Introduction to the Recommended Design Pattern
39.2 Other Approaches
39.3 Example
39.4 How to Invoke a Java Class from a SOA Composite Application
39.5 Securing the Design Pattern
39.6 Verifying the Deployment
39.7 Troubleshooting the Use Case
39.8 What You May Need to Know About Invoking Custom Java Code from a SOA Composite

40 Manage Tasks from an Oracle ADF Application
40.1 Introduction to the Recommended Pattern
40.2 Other Approaches
40.3 Example
40.4 How to Manage a Human Task Flow from an ADF Application
40.5 Other Approaches .......................................................................................................................... 40-5
40.6 Securing the Design Pattern ............................................................................................................. 40-6
40.7 Verifying the Deployment .................................................................................................................. 40-7
40.8 Troubleshoot the Use Case ................................................................................................................ 40-8
  40.8.1 Worklist Notification Locale Does Not Honor the Regional Applications Session Setting .. 40-8
  40.8.2 Task Does Not Display in Worklist Application ................................................................. 40-8
  40.8.3 Task Details Do Not Display in the ADF Task Flow .............................................................. 40-9
  40.8.4 Logging ....................................................................................................................................... 40-10
40.9 What You May Need to Know About Managing Tasks from an ADF Application ................. 40-11

41 Work with Data from a Remote ADF Business Components Service

41.1 Introduction to the Recommended Design Pattern ...................................................................... 41-1
41.2 Potential Approaches ....................................................................................................................... 41-1
41.3 Example ........................................................................................................................................ 41-2
41.4 How to Create Service-Based Entity Objects and View Objects .................................................... 41-2
41.5 Securing the Design Pattern ........................................................................................................... 41-3
41.6 Verifying the Deployment ............................................................................................................... 41-3
41.7 Troubleshooting the Use Case ......................................................................................................... 41-3
41.8 Understanding the Transactional Behavior of Service-Based Entity Objects and View Objects ......... 41-3
41.9 Known Issues and Workarounds .................................................................................................... 41-4

42 Invokie an Asynchronous Service from a SOA Composite

42.1 Introduction to the Recommended Design Pattern ...................................................................... 42-1
42.2 Other Approaches ............................................................................................................................. 42-2
42.3 Example ........................................................................................................................................ 42-2
42.4 How to Invoke a SOA Composite Application from Within a SOA Composite Application ......... 42-3
  42.4.1 Define a New Web Service Reference ..................................................................................... 42-3
  42.4.2 Wiring the BPEL Process to the New Web Service Reference ............................................... 42-4
  42.4.3 Invoke the Asynchronous Web Service from the BPEL Flow ................................................. 42-6
  42.4.4 What Happens When You Invoke an Asynchronous Service from within a SOA Composite Application .............................................................................................................. 42-12
  42.4.5 What Happens at Runtime: How an Asynchronous Service is Invoked from within a SOA Composite Application .............................................................................................................. 42-12
42.5 Securing the Design Pattern ........................................................................................................... 42-12
42.6 Verifying the Deployment ............................................................................................................... 42-12
42.7 Troubleshoot the Use Case ............................................................................................................... 42-13
  42.7.1 Deployment .............................................................................................................................. 42-13
  42.7.2 Runtime ..................................................................................................................................... 42-13
42.8 What You May Need to Know About Invoking an Asynchronous Service from Another SOA Composite ............................................................................................................................... 42-13
43  Synchronously Invoking an ADF Business Components Service from an Oracle ADF Application

43.1  Introduction to the Recommended Design Pattern
43.2  Potential Approaches
43.3  Example
43.4  How to Invoke an ADF Business Components Service from an Oracle ADF Application
43.5  Securing the Design Pattern
43.6  Verifying the Deployment

44  Implement an Asynchronous Service Initiation with Dynamic UI Update

44.1  Introduction to the Recommended Design Pattern
44.2  Potential Approaches
44.3  Example
44.4  How to Implement an Asynchronous Service Initiation with Dynamic UI Update
44.4.1  Writing the Active Data Handler
44.4.2  Building the Supporting Active Data Entry Classes
44.4.3  Register the Active Data Collection Model with the Oracle ADF UI Page
44.4.4  Register the Component Managed JavaBean for Supporting Method Actions
44.4.5  Reference the Managed JavaBean in the Page UI
44.4.6  Create the Data Model and Add Application Module Methods
44.4.7  Create a SOA Composite that Subscribes to the Published Event
44.4.8  Construct a BPEL Process to Perform Asynchronous Work
44.4.9  Invoking the ADF Business Components Service
44.5  Securing the Design Pattern
44.6  Verifying the Deployment
44.7  Troubleshooting the Use Case
44.8  What You May Need to Know About Initiating an Asynchronous Service with Dynamic UI Update
44.9  Known Issues and Workarounds

45  Manage Tasks Programmatically

45.1  Introduction to the Recommended Design Pattern
45.2  Potential Approaches
45.3  Example
45.4  Manage Human Workflow Tasks from a Java Application
45.4.1  How to Connect to the Task Service/Task Query Service
45.4.2  How to Use the Single Server Task Service API
45.4.3  How to Use the Single Server Task Query Service API
45.4.4  How to Use the Federated Server Task Query Service API
45.4.5  How to Query and Traverse Federated and Non-federated Query Result Sets
45.5  Other Approaches
45.6  Securing the Design Pattern
47.6 Verify the Deployment
47.6.1 How to Verify the Deployment of the XFamilyPub Composite
47.6.2 How to Verify the Deployment of the XFamilySub Composite
47.7 Troubleshoot the Use Case
47.7.1 Privileges to FUSION_RUNTIME
47.7.2 Aqueue enabled for Enqueuing and Dequeuing
47.7.3 AQ_INVALID_QUEUE_TYPE

48 Use Oracle Enterprise Scheduler to Invoke an Asynchronous Service
48.1 Introduction
48.2 Alternative Approaches
48.3 Example
48.4 Invoking an Asynchronous ADF Business Components Service from an Oracle Enterprise Scheduler Java Job
48.5 Securing the Design Pattern
48.6 Verifying the Design Pattern
48.7 Troubleshooting the Use Case
48.8 What You May Need to Know About the Design Pattern

49 Get Started with Security
49.1 Introduction to Securing Oracle Fusion Applications
49.1.1 Architecture
49.1.2 Authentication
49.1.3 Authorization
49.2 Authentication Techniques and Best Practices
49.2.1 API Support for Authentication
49.2.2 Expression Language Support for Authentication
49.2.3 Non-browser Based Login
49.3 Authorization Techniques and Best Practices
49.3.1 Function Security
49.3.2 Data Security
49.4 Learning More About Technologies Used to Secure Oracle Fusion Applications

50 Implement Oracle Fusion Data Security
50.1 Introduction to Oracle Fusion Data Security
50.1.1 Terminology
50.1.2 Integrating Oracle Fusion Data Security with Oracle Platform Security Services (OPSS)
50.1.3 Integrating Data Security Task Flows into Oracle Fusion Functional Setup Manager
50.1.4 Integrating Oracle Fusion Data Security with User Sessions
50.1.5 Integrating Oracle Fusion Data Security with Virtual Private Database (VPD)
50.2 Managing Data Security Artifacts in the Oracle Fusion Data Security Policy Tables
50.2.1 How to Get Started Managing Data Security ................................................................. 50-8
50.2.2 What You May Need to Know About Administering Oracle Fusion Data Security Policy Tables .................................................................................................................................................... 50-9
50.3 Integrating with ADF Business Components .................................................................... 50-9
  50.3.1 How to Configure the ADF Data Model Project ......................................................... 50-10
  50.3.2 How to Secure Rows for Read, Update and Delete Access ........................................ 50-11
  50.3.3 How to Secure Rows for Custom Operations and Specific View Objects .................. 50-11
  50.3.4 What You May Need to Know About Naming Actions .............................................. 50-14
  50.3.5 What You May Need to Know About Column-Level Security .................................. 50-14
  50.3.6 What You May Need to Know About Translatable Entity Objects ............................. 50-14
  50.3.7 What Happens at Runtime: How Oracle Fusion Data Security Filters View Instance Rows .......................................................................................................................................................................................... 50-15
  50.3.8 How to Perform Authorization Checks for Custom Operations ............................... 50-15
  50.3.9 How to Test Permissions Using Expression Language Expressions in the User Interface ............................................................................................................................................ 50-16
50.4 Use Oracle Fusion Data Security to Grant Access to Secured Business Objects .............. 50-18
  50.4.1 How to Create Data Security Policies for Custom Business Objects ......................... 50-20
  50.4.2 How to Use Parameterized Conditions When Securing a Business Object ................ 50-25
  50.4.3 How to Create Test Users in JDeveloper .................................................................... 50-28
  50.4.4 What You May Need to Know About Creating Application Roles ............................ 50-28
50.5 Getting Security Information from the Application User Session Context ....................... 50-29
  50.5.1 How to Use the DataSecurityAM API to Get Session Context Information ................ 50-29
  50.5.2 How to Use the PL/SQL Data Security API to Check User Privileges ....................... 50-30
50.6 Understanding Data Security Performance Best Practices ............................................. 50-32
50.7 Validating Data Security with Diagnostic Scripts ................................................................ 50-33
  50.7.1 How to Validate Data Security Configuration with Diagnostic Scripts ..................... 50-34
  50.7.2 How to Validate Applications Context ....................................................................... 50-35
50.8 Integrating with Data Security Task Flows ...................................................................... 50-37
  50.8.1 About Integrating the Data Security Task Flows into Your Application .................... 50-37
  50.8.2 How to Configure Data Security Task Flows to Display in the Primary Window .......... 50-41
  50.8.3 How to Configure the Object Instance Task Flow to Display in a Dialog ................... 50-47
  50.8.4 How to Grant the Data Security Task Flows Access to the Application Policy Store .......................................................................................................................................................................................... 50-53
  50.8.5 How to Grant the End User Access to the Data Security Task Flows .......................... 50-53
  50.8.6 How to Map the Application to an Existing Application Stripe ............................... 50-54

51 Implement Application User Sessions .................................................................................. 51-1
  51.1 Introduction to Application User Sessions ....................................................................... 51-1
  51.2 Configuring Your Project to Use Application User Sessions ......................................... 51-2
    51.2.1 How to Configure Your Project to Use Application User Sessions ........................ 51-2
    51.2.2 How to Configure the ADF Business Component Browser ..................................... 51-3
    51.2.3 How to Use the ApplSession Logger for Troubleshooting ...................................... 51-3
    51.2.4 What Happens at Runtime: How the Application User Session is Used ................ 51-4
## Design and Secure View Objects for Oracle Business Intelligence Applications

### 62.1 Introduction to View Objects for Oracle Business Intelligence Applications

- About Entity Object Guidelines
- About Association Guidelines
- About View Object Guidelines
- Understanding View Links Guidelines
- Understanding View Criteria Guidelines

### 62.2 Understanding General Design Guidelines

- About Entity Object Guidelines
- About Association Guidelines
- About View Object Guidelines
- Understanding View Links Guidelines
- Understanding View Criteria Guidelines

### 62.3 Understanding Oracle Business Intelligence Design Patterns

- Understanding Flattened View Objects
- Understanding Fact-Dimension Relationships
- Understanding Self Referencing Entities (Self-Joins)
- Understanding Business Intelligence Filters
- Understanding Translations
- Understanding Date Effectivity

### 62.4 About Designing and Securing Fact View Objects

- About Designing Fact View Objects
- About Securing Fact View Objects

### 62.5 About Designing and Securing Dimension View Objects

- Overview of Designing Dimension View Objects
- About Designing Business Unit Dimensions
- About Securing Dimension View Objects
- About Using Multi-Valued Dimension Attributes
- About Using Junk Dimensions and Mini Dimensions
- About Using Secured and Unsecured Dimension View Objects

### 62.6 About Designing Date Dimensions

- About Using the Gregorian Calendar
- About Using the Fiscal Calendar
- About Using the Projects Calendar
- About Using Timestamp Columns
- About Using Role-Playing Date Dimensions

### 62.7 About Designing Lookups as Dimensions

- About Securing Data on Lookups

### 62.8 About Designing and Securing Tree Data

- About Designing a Column-Flattened View Object for Oracle Business Intelligence
- About Customizing the FND Table Structure and Indexes
- About Using Declarative SQL Mode to Design View Objects for Oracle Business Intelligence Applications
- Guidelines for ATG-Registration and BICVO Generation
- Guidelines for Hierarchy Depth and Conformance
62.8.6 About Securing ADF Business Components View Objects for Trees .......................... 62-28
62.9 About Supporting Flexfields for Oracle Business Intelligence ....................................... 62-30
62.10 About Supporting SetID ...................................................................................................... 62-31
  62.10.1 How to Expose the SetID Attribute for Set-Enabled Lookups .................................. 62-31
  62.10.2 How to Expose the SetID Attribute for Set-Enabled Reference Tables .................. 62-31
62.11 About Supporting Multi-Currency ..................................................................................... 62-32

63 Implement ADF Desktop Integration
  63.1 Oracle Application Development Framework Desktop Integration Standards and Guidelines ................................................................. 63-1
    63.1.1 How to Structure the ADF Desktop Integration Directories ................................ 63-2
    63.1.2 How to Name Your ADF Desktop Integration Files ........................................... 63-3
    63.1.3 How to Implement the Dialog Attributes Declarative Component .................... 63-4
  63.2 Skinning Excel ADF Desktop Integration Workbooks ....................................................... 63-6
  63.3 Configure the WebLogic Server Frontend ........................................................................ 63-6

64 Create Customizable Applications
  64.1 Introduction to Creating Customizable Applications ............................................................. 64-1
  64.2 Prepare an Application for Customizations .................................................................... 64-2
    64.2.1 How to Set Project Properties to Enable User and Seeded Customizations ........ 64-3
    64.2.2 How to Configure the Persistence Change Manager ............................................ 64-4
    64.2.3 How to Enable Translations of Customized Strings .............................................. 64-6
    64.2.4 How to Add Composer Technology Scope to Your Project ................................. 64-7
    64.2.5 How to Enable the User Customization of the UI Shell Template ....................... 64-7
    64.2.6 How to Create a Database Connection at the IDE Level ..................................... 64-9
  64.3 Enable Runtime Customization of Pages and Components ............................................. 64-9
    64.3.1 How to Enable Pages for Runtime Customization .............................................. 64-12
    64.3.2 How to Enable End-User Personalizations for a Page .......................................... 64-14
    64.3.3 How to Restrict Customization of a Page, Page Fragment, or Component .......... 64-14
    64.3.4 How to Control the Editing of Components in Page Composer ......................... 64-15
    64.3.5 How to Authorize the Runtime Customization of Pages and Task Flows ............ 64-17
    64.3.6 How to Persist Implicit Runtime Customizations ................................................. 64-18

65 Work with Extensions to Oracle Enterprise Scheduler
  65.1 Introduction to Oracle Enterprise Scheduler Extensions .................................................. 65-2
  65.2 Understanding Standards and Guidelines ...................................................................... 65-2
  65.3 About Creating and Implementing a Scheduled Job in Oracle JDeveloper .................... 65-2
    65.3.1 How to Create and Implement a Scheduled Job in JDeveloper .......................... 65-3
    65.3.2 What Happens When You Create and Implement a Scheduled Job in JDeveloper 65-3
    65.3.3 What Happens at Runtime: How a Scheduled Job Is Created and Implemented in JDeveloper ........................................................................ 65-3
  65.4 About Creating a Job Definition ....................................................................................... 65-4
    65.4.1 Understanding Job Definitions and Job Types ..................................................... 65-4
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>65.4.2</td>
<td>How to Define File Groups for a Job</td>
<td>65-9</td>
</tr>
<tr>
<td>65.4.3</td>
<td>What Happens When You Create a Job Definition</td>
<td>65-11</td>
</tr>
<tr>
<td>65.4.4</td>
<td>What Happens at Runtime: How Job Definitions Are Created</td>
<td>65-11</td>
</tr>
<tr>
<td>65.5</td>
<td>About Configuring a Spawned Job Environment</td>
<td>65-11</td>
</tr>
<tr>
<td>65.5.1</td>
<td>About Creating an Environment File for Spawned Jobs</td>
<td>65-11</td>
</tr>
<tr>
<td>65.5.2</td>
<td>How to Configure an Oracle Wallet for Spawned Jobs</td>
<td>65-13</td>
</tr>
<tr>
<td>65.5.3</td>
<td>About Migrating Spawned Job Environment Properties to the Production</td>
<td>65-14</td>
</tr>
<tr>
<td>65.5.4</td>
<td>What Happens When You Configure a Spawned Job Environment</td>
<td>65-18</td>
</tr>
<tr>
<td>65.6</td>
<td>About Implementing a PL/SQL Scheduled Job</td>
<td>65-18</td>
</tr>
<tr>
<td>65.6.1</td>
<td>Standards and Guidelines for Implementing a PL/SQL Scheduled Job</td>
<td>65-18</td>
</tr>
<tr>
<td>65.6.2</td>
<td>About Defining Metadata for a PL/SQL Scheduled Job</td>
<td>65-18</td>
</tr>
<tr>
<td>65.6.3</td>
<td>How to Implement a PL/SQL Scheduled Job</td>
<td>65-18</td>
</tr>
<tr>
<td>65.6.4</td>
<td>What Happens When You Implement a PL/SQL Job</td>
<td>65-18</td>
</tr>
<tr>
<td>65.6.5</td>
<td>What Happens at Runtime: How a PL/SQL Job is Implemented</td>
<td>65-20</td>
</tr>
<tr>
<td>65.7</td>
<td>About Implementing a SQL*Plus Scheduled Job</td>
<td>65-21</td>
</tr>
<tr>
<td>65.7.1</td>
<td>Standards and Guidelines for Implementing a SQL*Plus Scheduled Job</td>
<td>65-21</td>
</tr>
<tr>
<td>65.7.2</td>
<td>How to Implement a SQL*Plus Job</td>
<td>65-21</td>
</tr>
<tr>
<td>65.7.3</td>
<td>About Using the SQL*Plus Runtime API</td>
<td>65-21</td>
</tr>
<tr>
<td>65.7.4</td>
<td>What Happens When You Implement a SQL*Plus Job</td>
<td>65-22</td>
</tr>
<tr>
<td>65.7.5</td>
<td>What Happens at Runtime: How a SQL*Plus Job Is Implemented</td>
<td>65-23</td>
</tr>
<tr>
<td>65.8</td>
<td>About Implementing a SQL*Loader Scheduled Job</td>
<td>65-23</td>
</tr>
<tr>
<td>65.8.1</td>
<td>How to Implement a SQL*Loader Scheduled Job</td>
<td>65-23</td>
</tr>
<tr>
<td>65.8.2</td>
<td>What Happens When You Implement a SQL*Loader Scheduled Job</td>
<td>65-24</td>
</tr>
<tr>
<td>65.9</td>
<td>About Implementing a Perl Scheduled Job</td>
<td>65-25</td>
</tr>
<tr>
<td>65.9.1</td>
<td>How to Implement a Perl Scheduled Job</td>
<td>65-25</td>
</tr>
<tr>
<td>65.9.2</td>
<td>What Happens When You Implement a Perl Scheduled Job</td>
<td>65-26</td>
</tr>
<tr>
<td>65.10</td>
<td>About Implementing a C Scheduled Job</td>
<td>65-28</td>
</tr>
<tr>
<td>65.10.1</td>
<td>About Defining Metadata for a C Scheduled Job</td>
<td>65-28</td>
</tr>
<tr>
<td>65.10.2</td>
<td>How to Implement a C Scheduled Job</td>
<td>65-28</td>
</tr>
<tr>
<td>65.10.3</td>
<td>About the Scheduled C Job API</td>
<td>65-28</td>
</tr>
<tr>
<td>65.10.4</td>
<td>About Testing a C Scheduled Job</td>
<td>65-31</td>
</tr>
<tr>
<td>65.10.5</td>
<td>What Happens When You Implement a C Scheduled Job</td>
<td>65-31</td>
</tr>
<tr>
<td>65.10.6</td>
<td>What Happens at Runtime: How a C Scheduled Job Is Implemented</td>
<td>65-34</td>
</tr>
<tr>
<td>65.11</td>
<td>About Implementing a Host Script Scheduled Job</td>
<td>65-34</td>
</tr>
<tr>
<td>65.12</td>
<td>About Implementing a Java Scheduled Job</td>
<td>65-35</td>
</tr>
<tr>
<td>65.12.1</td>
<td>About Defining Metadata for a Scheduled Java Job</td>
<td>65-35</td>
</tr>
<tr>
<td>65.12.2</td>
<td>About Using the Java Runtime API</td>
<td>65-35</td>
</tr>
<tr>
<td>65.12.3</td>
<td>About Cancelling a Scheduled Java Job</td>
<td>65-35</td>
</tr>
<tr>
<td>65.12.4</td>
<td>What Happens at Runtime: How a Java Scheduled Job Is Implemented</td>
<td>65-36</td>
</tr>
<tr>
<td>65.13</td>
<td>About Elevating Access Privileges for a Scheduled Job</td>
<td>65-36</td>
</tr>
<tr>
<td>65.13.1</td>
<td>How to Elevate Access Privileges for a Scheduled Job</td>
<td>65-37</td>
</tr>
<tr>
<td>65.13.2</td>
<td>How Access Privileges Are Elevated for a Scheduled Job</td>
<td>65-38</td>
</tr>
</tbody>
</table>
65.13.3 What Happens When Access Privileges Are Elevated for a Scheduled Job
65.14 About Enabling Multiple Language Support for Jobs
65.14.1 About Defining Java Multiple Language Support Functions
65.14.2 About Defining PL/SQL Multiple Language Support Functions
65.14.3 How to Define ADF Business Components Multiple Language Support Functions
65.14.4 About Processing Output for Multiple Language Jobs
65.14.5 What Happens When You Enable Multiple Language Support for Jobs
65.14.6 What Happens at Runtime: How Multiple Language Support Is Enabled for Jobs
65.14.7 Standards and Guidelines Regarding Multiple Language Support Functions
65.15 About Creating an Oracle ADF User Interface for Submitting Job Requests
65.15.1 How to Create an Oracle ADF User Interface for Submitting Job Requests
65.15.2 How to Add a Custom Task Flow to an Oracle ADF User Interface for Submitting Job Requests
65.15.3 About Handling Schedule Request Submission and Jobset UI Submit and Cancel Button Events
65.15.4 How to Enable Support for Context-Sensitive Parameters in an Oracle ADF User Interface for Submitting Job Requests
65.15.5 How to Save and Schedule a Job Request Using an Oracle ADF UI
65.15.6 How to Submit a Job Using a Saved Schedule in an Oracle ADF UI
65.15.7 About Notifying Users of the Status of Executed Jobs
65.15.8 What Happens When You Create an Oracle ADF User Interface for Submitting Job Requests
65.15.9 What Happens at Runtime: How an Oracle ADF User Interface for Submitting Job Requests Is Created
65.16 Specify Optional Parameters
65.17 About Submitting Job Requests Using the Request Submission API
65.17.1 About Using the Java Request Submission API
65.17.2 About Using the PL/SQL Request Submission API
65.17.3 About Using the C Request Submission API
65.18 About Defining Oracle Business Intelligence Publisher Postprocessing Actions for a Scheduled Job
65.18.1 Perform Prerequisite Tasks
65.18.2 How to Define Oracle BI Publisher Postprocessing for a Scheduled Job
65.18.3 About Defining Oracle BI Publisher Postprocessing Actions for a Scheduled PL/SQL Job
65.18.4 About Invoking Postprocessing Actions Programmatically
65.18.5 What Happens When You Define Oracle BI Publisher Postprocessing Actions for a Scheduled Job
65.18.6 What Happens at Runtime: How Oracle BI Publisher Postprocessing Actions are Defined for a Scheduled Job
65.19 About Monitoring Scheduled Job Requests Using an Oracle ADF UI
65.19.1 How to Monitor Scheduled Job Requests ................................................................. 65-69
65.19.2 How to Embed a Table of Search Results as a Region on a Page ......................... 65-71
65.19.3 How to Log Scheduled Job Requests in an Oracle ADF UI ................................ 65-72
65.19.4 Tips for Troubleshooting an Oracle ADF UI Used to Monitor Scheduled Job Requests ........................................................................................................ 65-73
65.20 About Integrating Data Security Task Flows into Applications Setup Manager ........ 65-75
65.21 About Using a Task Flow Template for Submitting Scheduled Requests Through an Oracle ADF UI ................................................................................................................ 65-76
  65.21.1 About Using a Task Flow Template for Submitting Scheduled Requests through an Oracle ADF UI .......................................................... 65-76
  65.21.2 How to Extend the Task Flow Template for Submitting Scheduled Requests through an Oracle ADF UI ................................................. 65-77
  65.21.3 What Happens When you Use a Task Flow Template for Submitting Scheduled Requests through an Oracle ADF UI ...................................... 65-78
  65.21.4 What Happens at Runtime: How a Task Flow Template Is Used to Submit Scheduled Requests through an Oracle ADF UI...................... 65-78
65.22 About Securing Oracle ADF UIs .................................................................................... 65-78
65.23 About Integrating Scheduled Job Logging with Oracle Fusion Applications ............... 65-79
65.24 About Logging Scheduled Jobs and Writing to Output Files ....................................... 65-79
  65.24.1 About Using the Request Log ................................................................................. 65-80
  65.24.2 About Using the Output File ................................................................................. 65-80
  65.24.3 About Debugging and Error Logging ................................................................. 65-81

66 Oracle Enterprise Scheduler Security

66.1 Introduction to Oracle Enterprise Scheduler Security .................................................. 66-1
  66.1.1 Oracle Enterprise Scheduler Metadata Access Control ......................................... 66-1
  66.1.2 Oracle Enterprise Scheduler Job Execution Security .............................................. 66-2
66.2 Configuring Metadata Security for Oracle Enterprise Scheduler .................................. 66-2
  66.2.1 How to Enable Application Security with Oracle ADF Security Wizard ................ 66-3
  66.2.2 How to Define Principals for Security .................................................................... 66-4
  66.2.3 How to Create Grants with Oracle Enterprise Scheduler Metadata Pages ............... 66-5
  66.2.4 How to Create Grants with Oracle ADF Security Wizard ......................................... 66-6
  66.2.5 MetadataPermission APIs ...................................................................................... 66-8
  66.2.6 What Happens When You Configure Metadata Security ........................................ 66-9
  66.2.7 Related Links ......................................................................................................... 66-9
66.3 Configure Web Service Security for Oracle Enterprise Scheduler .............................. 66-9
66.4 Configure PL/SQL Job Security for Oracle Enterprise Scheduler ................................. 66-9
66.5 Elevating Privileges for Oracle Enterprise Scheduler Jobs .......................................... 66-9
66.6 Configuring a Single Policy Stripe in Oracle Enterprise Scheduler ............................ 66-10
  66.6.1 How to Configure a Single Policy Stripe in Oracle Enterprise Scheduler .................. 66-10
  66.6.2 What Happens When You Configure a Single Policy Stripe .................................... 66-11
  66.6.3 What Happens at Runtime ...................................................................................... 66-12
66.7 Configure Oracle Fusion Data Security for Job Requests .............................................. 66-12
Abstract

Documentation for external Oracle Fusion Applications developers that describes Oracle Fusion Middleware components, installing JDeveloper, deploying applications on WebLogic Server (WLS), using Applications Core Technology (ApplCore), customization, security, Flexfields, developing web applications with the UI Shell page template and patterns, Enterprise Crawl and Search (ECSF), database schema deployment, seed data, and use cases.
Welcome to the *Oracle Fusion Applications Developer’s Guide!* This guide describes the Oracle Middleware Extensions for Applications for developing Oracle Fusion Applications using the Oracle Fusion Middleware components. This guide includes guidelines on how to set up your development environment and build, test, and deploy Oracle Fusion Applications. It includes specific feature details needed by developers when using the Oracle Middleware Extensions to create applications.

In this user guide, the nomenclature “11.12.x.0.0”, where “x” is a number, is used to indicate the release and patch releases for which the guide is applicable. When using this document, be sure to replace “x” with the number of the release that is being used.

**Audience**

This document is intended for Oracle Fusion Applications Developers and assumes familiarity with:

- Java and SQL
- Java EE, Java Servlets, and JavaServer Pages
- Oracle Application Development Framework
- Oracle JDeveloper

**Documentation Accessibility**


**Access to Oracle Support**

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit [http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info](http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info) or visit [http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs](http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs) if you are hearing impaired.

**Related Documents**

For more information, see the following documentation:

- *Oracle Fusion Applications Extensibility Guide for Developers*
Oracle Fusion Middleware Developing Fusion Web Applications with Oracle Application Development Framework

Oracle Fusion Middleware Developing Web User Interfaces with Oracle ADF Faces

Oracle Fusion Middleware Developing Oracle ADF Mobile Browser Applications

Oracle Fusion Middleware Developing Applications with Oracle ADF Desktop Integration

Oracle Fusion Middleware Developer’s Guide for Oracle Business Intelligence Enterprise Edition

Oracle Fusion Middleware Metadata Repository Builder’s Guide for Oracle Business Intelligence Enterprise Edition

Oracle Fusion Middleware Developer’s Guide for Oracle Business Intelligence Publisher

Oracle Fusion Middleware Report Designer’s Guide for Oracle Business Intelligence Publisher

Oracle Fusion Middleware Developing Business Processes with Oracle Business Process Management Studio

Oracle Fusion Middleware Developing Business Processes with Oracle Business Process Composer

Oracle Fusion Middleware Developing Integration Projects with Oracle Data Integrator

Oracle Fusion Middleware Connectivity and Knowledge Modules Guide for Oracle Data Integrator

Oracle Fusion Middleware Developing Knowledge Modules with Oracle Data Integrator

Oracle Fusion Middleware Developing Oracle WebCenter Content: Imaging

Oracle Fusion Middleware Application Developer’s Guide for Oracle Identity Management

Oracle Fusion Middleware Developing SOA Applications with Oracle SOA Suite

Oracle Fusion Middleware Designing Business Rules with Oracle Business Process Management


Oracle Fusion Middleware Understanding Technology Adapters

Oracle Fusion Middleware Monitoring Business Activity with Oracle BAM

Oracle Fusion Middleware Developing WebCenter Portal Assets and Custom Components with Oracle JDeveloper

Oracle Fusion Middleware Using Oracle WebCenter Portal

Oracle WebLogic Communication Services Developer’s Guide

Oracle Fusion Middleware Securing Applications with Oracle Platform Security Services
# Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><strong>monospace</strong></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
The following topics introduce the new and changed features of the Oracle Fusion Applications Developer’s Guide Release 12 (11.12.x.0.0) and other significant changes that are described in this guide, and provides pointers to additional information.

New and Changed Features for Release 12 (11.12.x.0.0)

The Oracle Fusion Applications Developer’s Guide Release 12 (11.12.x.0.0) includes the following new and changed features for this document.

- The nomenclature “11.12.x.0.0”, where “x” is a number, is used to indicate the release and patch releases for which the guide is applicable.
- The Implementing Attachments chapter has been extensively reorganized and updated.
- A new subsection, About Configuring the Attachment Link Component UI, has been added to the Configuring Attachments Components Properties section.

Other Significant Changes in this Document for Release 12 (11.12.x.0.0)

For Release 12 (11.12.x.0.0), this guide has been updated in several ways. Following are the sections that have been added or changed.

- Implementing Search Functions in the UI Shell has received numerous updates.
1. Get Started with Oracle Fusion Applications

This chapter describes how to design and build your Oracle Fusion Applications using Oracle standards and guidelines. It includes an overview of Oracle Fusion technologies and using Oracle Application Development Framework (ADF) functional patterns.

This chapter includes this section:

• Overview of Fusion Technologies

1.1 Overview of Fusion Technologies

Oracle Fusion web applications are a set of business-related applications developed with the help of various technologies. This section describes the various technologies with which an Oracle Fusion web application developer works when developing the applications.

The following is a list of the various categories of technologies that, as an Oracle Fusion web application developer, you will encounter. This section does not go into the details about why the specified technologies have been chosen, the main intention is to give you an overview of the various technologies that are used to develop Oracle Fusion web applications.

• User interface (UI) technologies
• Model technologies
• Backend technologies
• Orchestration technologies
• Security
• Customization-related technologies
• Metadata services
• General middle-tier technologies
• Application server technologies

UI Technologies

Technologies that are used to create user interfaces fall into this category. The technologies that must be used in Oracle Fusion to create these user interfaces are:

• ADF Faces Rich Client:
  The ADF Faces rich client technology is used to create browser-based user interfaces. It provides a set of UI components, which can be dragged and dropped...
to create UIs. Among these ADF components are other components called the data visualization tools, which are a set of rich interactive components that provide graphical and tabular capabilities for visualizing and analyzing data.

For more information about ADF Faces, see the Oracle Fusion Middleware Developing Fusion Web Applications with Oracle Application Development Framework guide.

For more information about ADF Faces rich client components, see the Oracle Fusion Middleware Developing Web User Interfaces with Oracle ADF Faces guide.

• ADF Desktop Integration:
This technology is used to create interfaces accessed through Microsoft Excel.

For more information about ADF Desktop Integration, see the Oracle Fusion Middleware Developing Applications with Oracle ADF Desktop Integration guide.

• ADF Mobile:
Oracle ADF Mobile helps developers build and extend enterprise applications for iOS and Android from a single code base. Based on a hybrid mobile architecture, ADF Mobile supports access to native device services, enables offline applications and protects enterprise investments from future technology shifts.

For more information about Oracle ADF Mobile, see the Oracle Fusion Middleware Developing Oracle ADF Mobile Browser Applications guide.

Model Technologies
Technologies that are used to represent the business logic and the data on which the business logic is based fall into this category. The UI technologies discussed previously can be based on any model technology such as Enterprise JavaBeans (EJB), Oracle Toplink, and so on. In Oracle Fusion, ADF Business Components is the model technology that is used in all applications.

Backend Technologies
These technologies are the set of storage technologies that are used to store the transactional and relational data. The primary technologies used in Oracle Fusion to store and retrieve data are:

• Oracle Database: This is used to store and retrieve all transactional and reference data.
  For more information see the Oracle Database Administrator’s Guide.

• Oracle Enterprise Scheduler: This is used to manage multi-dimensional data. Essbase provides adaptable data storage mechanisms for specific types of analytic and performance management applications. It is used to manage multi-dimensional data.

Orchestration Technologies
These are the technologies that are used in the service-oriented architecture (SOA) world. The primary purpose of these technologies is to assemble various services together to provide comprehensive functionality.

In Oracle Fusion, many product applications provide their functionality in the form of web services. Oracle Application Server BPEL Process Manager is used to assemble these web services together to provide end-to-end functionality.
For more information about SOA, see the *Developing SOA Applications with Oracle SOA Suite* guide.

**Security**

Security is an integral part of all of the technologies previously mentioned. The technology used to provide security for Oracle Fusion Applications is Oracle Platform Security Services (OPSS).

For more information about OPSS, see the *Oracle Fusion Middleware Oracle Platform Security Services (OPSS) & Oracle Authorization Policy Manager (OAPM) Frequently Asked Questions*.

**Customization-Related Technologies**

Customization-related technologies give customers the tools they need to customize the artifacts that developers have created. For example, the customer requires more information on the Invoices Entry UI that the developer created. They want to customize the UI by adding this extra information. To perform this type of customization, Metadata Services (MDS) technology is used.

Another level of customization, which is used to customize the UI pages at runtime, is called Design Time at Runtime (DTRT) customization. This type of customization is performed using the WebCenter technologies. (This uses Oracle Metadata Services (MDS) internally.)

**Additional Technologies**

In addition to the technologies previously discussed, there are many others that Oracle Fusion web application developers may encounter. These include:

- **Oracle Enterprise Scheduler Service**: Oracle Enterprise Scheduler Service provides the ability to run different Job Types, including: Java, PL/SQL, and Binary Scripts, distributed across the nodes in an Oracle Application Server Cluster. Oracle Enterprise Scheduler Service runs these jobs securely, with high availability and scalability, with load balancing and provides monitoring and management through Oracle Enterprise Manager Fusion Middleware Control.

  For more information about Oracle Enterprise Scheduler Service, see the *Developing Applications for Oracle Enterprise Scheduler* guide.

- **Oracle Enterprise Crawl and Search Framework (ECSF)**: ECSF helps expose application context information on business objects to enable full-text transactional search.

  For more information about Oracle Enterprise Crawl and Search Framework, see *Getting Started with Oracle Enterprise Crawl and Search Framework*.

- **Oracle Business Rules (OBR)**: Oracle Business Rules enable dynamic decisions at runtime allowing you to automate policies, computations, and reasoning while separating rule logic from underlying application code. This allows more agile rule maintenance and empowers business analysts with the ability to modify rule logic without programmer assistance and without interrupting business processes.

  For more information about Oracle Business Rules, see the *Designing Business Rules with Oracle Business Process Management* guide.
• **Oracle Data Integrator (ODI):** Oracle Data Integrator is a comprehensive data integration platform that covers all data integration requirements - from high-volume, high-performance batches, to event-driven, trickle-feed integration processes, to SOA-enabled data services.

For more information about Oracle Data Integrator, see the *Oracle Fusion Middleware Developing Integration Projects with Oracle Data Integrator* guide.
This chapter describes how to configure and test your development environment. It includes the steps for setting up your JDeveloper environment and Oracle Application Development Framework (Oracle ADF) installation, running and deploying applications on Integrated WebLogic Server, and the basic steps for setting up your Oracle SOA Suite development environment.

This chapter includes these sections:

- Introduction to the Development Environment
- Setting Up the JDeveloper-based Personal Environment
- Improving Shared and Personal Environments Performance
- Using Deployment Profiles Settings
- Configuring the Oracle Enterprise Scheduler (ESS)
- Testing Your Installation
- Using Best Practices for Setting Up the Development Environment

### 2.1 Introduction to the Development Environment

**Note:**

Although JDeveloper works well in a 32-bit environment, this chapter assumes that you are using a 64-bit operating system and a 64-bit JDK.

If you do need to create and run Standalone WebLogic Server on a local PC, such as when doing SOA work, then the operating system must be 64-bit. The MW_HOME directory that is installed contains several Oracle Homes and the Oracle Business Intelligence installation is 64-bit only.

Also, a 32-bit operating system is limited to accessing 4GB of RAM. 8GB of RAM is preferred.

Oracle Fusion Applications provisioning involves installing, patching, configuring, and deploying all the enterprise components. At the end of the provisioning process, the system will be operational. An application administrator will be able to log in to the application and begin the process of configuring the functional (application specific) components.

On-site, the administrator uses eDelivery or the DVD media to kick-start the provisioning processes to create the test environment and the production environment. These environments are completely isolated from one another and set
up in an identical manner. There is no reuse of, for example, the database from the production environment in the test environment, or reuse of the Identity Store across the environments.

These environments need to be extremely stable and should not be affected by development projects. Typical development projects include creating new customizations for existing Oracle Fusion applications, developing new in-house Oracle Fusion applications, and extending Oracle Fusion applications with additional functionality. These development projects will typically involve a team of developers that needs to reuse certain parts, but still needs the isolation to run, test, and debug without affecting other team members.

As a developer, you will work with one development environment that has two parts:

- **Shared environment**
  This fully-provisioned Oracle Fusion Applications environment includes the database and LDAP. It also can host the instance of WebLogic Server if you require SOA for your customization work. The shared environment usually is designated as the Test environment.

- **Personal environment**
  This is the JDeveloper instance and an instance of Integrated WebLogic Server that can run on your local desktop or laptop. You still would connect to the database and LDAP instances on the shared environment.

---

**Note:**

To use Integrated WebLogic Server, the connection to a database and an LDAP server is required. The database needs all the correct schemas for Oracle Fusion applications and a properly-seeded LDAP.

The Personal environment also can have a standalone MW_HOME and WebLogic Server domain, such as:

- **standalone:** Only the AdminServer where you can run and deploy ADF applications.
- **adminsoa:** AdminServer plus SOA managed server.
- **adminall:** AdminServer configured for a standalone domain and a SOA managed server.
- **adminadfess:** AdminServer configured for standalone domain and an ESS managed server.

The shared environment plus the personal environment form the complete development environment.

### 2.1.1 Shared Environment

The shared environment is a fully-provisioned Oracle Fusion Applications environment to be shared among multiple developers. It normally will be set up by an administrator using the provisioning framework. This environment normally is set up on a machine that is more powerful than the normal developer’s machine, which often is a laptop.
Developers would obtain the EAR files to be customized from the filesystem from this environment.

The database, WebLogic Servers and common services provided by this environment would be used by the developers’ applications.

When it is provisioned, the shared environment contains:

- Database
- Oracle Middleware Home
- WebLogic Server domains
- Identity Store and Policy Store

Properties and features of the shared environment include:

- The shared environment can be accessed from the personal environment (see Personal Environment) so developers can reach the installation directory that normally is called MW_HOME.

- The exploded EAR directory in Middleware home can be opened from JDeveloper and a workspace can be created. The exploded EARs in Middleware home have the connections.xml and adf-config.xml files set up correctly to point to the correct database and other deployed applications in the shared environment.

- The LDAP and OPSS credentials in the personal environment point to the Identity Store and the Policy Store in the shared environment.

- WebLogic Server domains run in the shared environment and Oracle Fusion applications are deployed to those domains.

  The shared environment is a completely-provisioned Oracle Fusion Applications instance.

For more information about provisioning an environment, see "Provisioning a New Applications Environment" in the Installation Guide, and the Oracle Fusion Applications release notes.

2.1.2 Personal Environment

Each developer has a personal environment that uses the database, Middleware home, and the Identity and Policy stores from the shared environment. The shared environment is made available by using NFS mount or a mapped drive in the personal environment. In this environment, developers can use JDeveloper to run, test, and debug their changes. If network sharing is not an option, exploded EAR files can be copied to a local directory of the personal environment.

The personal environment consists of two parts:

- The JDeveloper-based environment that is created by installing JDeveloper, extension bundles and patches.
- The environment for Standalone WebLogic Server that is created using scripts and installer files.

Manually Deploying the oracle.apps.common.resource Shared Library

When an Integrated or a Standalone WebLogic Server instance has been launched, you will need to use the WebLogic Server Console to deploy
oracle.apps.common.resource.ear as a shared library using
oracle.apps.common.resource as the name. The file is located in the APPLTOP/
fusionapps/applications/com/deploy/ path of the fully-provisioned Oracle Fusion
Applications installation on the central server. See Managing Integrated WebLogic
Server.

If you forget to deploy that shared library, you will get an error when you try to run
any Oracle Fusion application in your Integrated or Standalone WebLogic Server
because they all have dependencies on this shared library.

## 2.2 Set Up the JDeveloper-based Personal Environment

You assemble this environment on your machine by performing these steps in this order:

- Install JDeveloper
- Install Extension Bundles
- Apply patches

Using JDeveloper and the Oracle Fusion Domain wizard, you can create the
fusion_apps_wls.properties file and a credential store. You will enter the host,
port, and other details for the database and Identity Store from the shared
environment. Eventually, the wizard will do the following:

- Create DefaultDomain.
- Extend DefaultDomain with WebLogic Server templates so that the shared
  libraries are added to the CLASSPATH and the system properties are set.
- Configure DefaultDomain with datasources and the Identity Store that are
  available in the shared environment.

DefaultDomain is run as part of Integrated WebLogic Server from within JDeveloper.
To create customizations for a shipped Oracle Fusion application, you can use
JDeveloper to point to the exploded EAR directory of the application in the shared
environment's Middleware home. A customization workspace will be created in
JDeveloper with the adf-config.xml file being modified such that the Metadata
Service (MDS) metadata store points to the filesystem. However, the DefaultDomain is
configured with the ApplicationDB datasource that points to the fusion runtime
schema that is installed in the database from the shared environment. Since the MDS
namespace has been altered while incorporating the exploded EAR directory, the
customizations that are created on the filesystem are picked up when the application is
run within Integrated WebLogic Server.

## 2.2.1 Before You Begin

Before you can use JDeveloper, there are several things you need to do.

### 2.2.1.1 Remove the SCIM Process

**Note:**

This step is not applicable if you are running a Windows environment.
If you are using your own workstation, you probably have a process called SCIM (Smart Common Input Method) running. SCIM is an input method platform used for multilingual work.

This process may prevent you from entering a password in the Oracle Fusion Domain wizard or anywhere a JPasswordField occurs. If it does interfere with your work, you can remove SCIM from your system by executing this command.

```
sudo yum remove scim
```

If you do not want to remove it completely, you can just kill the processes by executing the following command. However, if you just kill the processes instead of removing SCIM, you must kill them each time you reboot your system.

```
ps -ef | grep -i scim
```

You then can kill -9 all those processes.

### 2.2.1.2 Increase Open File Limit on Linux Machines

This system configuration change is required on Linux machines to increase the open file limit and resolve several JDeveloper, WebLogic Server and other "Too many files" errors when doing a build.

1. Add these two instructions to the `/etc/security/limits.conf` file:

   * soft nofile 8192
   * hard nofile 8192

   Note the leading asterisk (*).

2. Restart the machine to have these values take effect.

3. To check whether the settings took effect, change to a bash shell and run this command.

   ```
   [userid@blah ~]  bash
   bash-3.1$ > ulimit -n
   8192
   ```

### 2.2.1.3 Installing JDeveloper

Use the Oracle eDelivery system, download the Oracle JDeveloper 11g and Oracle Application Development Framework 11g and Oracle Fusion Applications Companion ZIP files that are appropriate for your operating system. Make sure to follow the instructions in the Readme file (click Readme at the top of the page).

**Note:**

*Always* use the Oracle eDelivery system. *Do not* download a random version from Oracle Technology Network (OTN).

Your administrator should make the contents of the ZIP files available on a shared directory. We strongly recommend that the administrator make the contents of the Oracle Fusion Applications Companion ZIP file available on a shared directory.
Follow these two basic steps to install JDeveloper.

- If you have not already done so, you must download and install the 64-bit JDK. To avoid the possibility of script errors, do not install to a directory location containing spaces, such as Program Files.

When installing JDeveloper, you will select the 64-bit JDK location you just installed.

If a 32-bit JDK was already installed and used, you must delete the old %MW_HOME% installation and the system directory in %JDEV_USER_HOME% before reinstalling JDeveloper and the Oracle Fusion Applications extensions updates.

- Install JDeveloper from the top level of the directory into which you extracted the ZIP files you downloaded from Oracle eDelivery.

For Windows, you can use the jdevstudioxxxxxinstall.exe installer (where xxxx is the version number). For Linux, you can use the jdevstudioxxxxxinstall.bin installer. If you decide to use the generic jdevstudioxxxxxinstall.jar installer, you must first install JDK 6 Update 24 from the Oracle Technology Network and then install JDeveloper using the generic installer.

The installation will let you specify the directory into which to install JDeveloper. To avoid possible errors when the installation scripts run, do not include any spaces in the path. This installation directory is referred to as MW_HOME.

When you have started JDeveloper, you will need to install the extension bundles from the fusion_apps_extensions directory in the location in which you extracted the Oracle Fusion Applications Companion files. See Adding Customization Extension Bundles to the jdev.conf File and Setting Up the JDeveloper-based Development Environment.

Related Links
The following document provides additional information related to subjects discussed in this section:

- For more installation information, see the Installing Oracle JDeveloper.

2.2.1.4 Add Customization Extension Bundles to the jdev.conf File

You must set this option before starting to customize an application if your application contains product-specific customization classes.
In $MW_HOME/jdeveloper/jdev/bin, open the jdev.conf file in a text editor and add this line:

```
AddVMOption -Dide.extension.extra.search.path /path/to/customization/bundles/directory1:/path/to/customization/bundles/directory2
```

where `-Dide.extension.extra.search.path /path/to/customization/bundles/directory1:/path/to/customization/bundles/directory2` is the fully-qualified path or paths to the directory or directories in which the JAR files containing the product-specific customization classes are located. Paths already exist as part of the provisioned environment.

- Check your adf-config.xml file to see if you are including any customization classes other than the ones described in "Personalizations Are Also Handled by the Customization Engine in MDS" in the Extensibility Guide for Developers and EnterpriseCC, which is deprecated. Those are your product-specific customization classes.

Example 2-1 shows customization class entries in the adf-config.xml file.

In Example 2-1, these two customization classes are outside the list:

```
<customization-class name="oracle.apps.hcm.common.core.HcmCountryCC"/>
<customization-class name="oracle.apps.hcm.common.core.HcmOrganizationCC"/>
```

- If you do have such product specific customization classes, you must modify your jdev.conf file to include the JAR files containing such customization classes. These JAR files have an Ext* prefix and are found under the application's APP_INF/lib directory.

- To find the path to such JAR files, contact your system administrator who can use these steps to locate APP_INF/lib/Ext*.jar:
  - Look under APP-INF/lib under the exploded EAR for all JAR files that start with Ext.
  - Find the JAR files that contain the product specific customization classes. These classes can be found in the adf-config file. If the product-specific customization class cannot be found in any of the JAR files under APP_INF/lib/Ext*.jar, look at all JAR files under the EarContents.

Example 2-1 Customization Class Entries in the adf-config.xml File

```
<cust-config>
  <match path="/*/">
    <customization-class name="oracle.apps.fnd.applcore.customization.GlobalCC"/>
    <customization-class name="oracle.apps.fnd.applcore.customization.SiteCC"/>
    <customization-class name="oracle.apps.fnd.applcore.customization.ProductCC"/>
    <customization-class name="oracle.apps.fnd.applcore.customization.EnterpriseCC"/>
    <customization-class name="oracle.apps.hcm.common.core.HcmCountryCC"/>
    <customization-class name="oracle.apps.hcm.common.core.HcmOrganizationCC"/>
    <customization-class name="oracle.apps.fnd.applcore.customization.UserCC"/>
  </match>
</cust-config>
```

2.2.1.5 Set Up the JDeveloper-based Development Environment

Follow these steps to create a development environment based on Integrated WebLogic Server:
1. From a command prompt, run `python -V` to see if you have Python 2.7.x on your machine. If you do not, install Python version 2.7.x. To avoid errors, do not install Python version 3.x. You can download Python from https://www.python.org.

2. Set these environment variables.

   **Note:**
   In many cases, it is convenient to have these in a script that is used to start JDeveloper. This is specially useful when you want to have different work directories for separate projects.

   **csh commands:**
   ```
   setenv PATH /path/to/python/bin:$PATH
   setenv HOME /path/to/user_dir (Important: If this variable is not set, the FADevConfigure.py script, which is called while creating an Oracle Fusion domain in JDeveloper, will fail.)
   setenv MW_HOME /path/to/JDeveloper/installation/directory
   setenv JAVA_HOME path/to/installed/64-bit/jdk (Important: A 32-bit JDK is installed into $MW_HOME when you install JDeveloper. If you are running the recommended 64-bit operating system, you will need to install a 64-bit JDK and set the JAVA_HOME variable to it.)
   setenv PATH $JAVA_HOME/bin:$PATH
   setenv JDEV_USER_HOME /path/to/a/directory (Optional) setenv FADEV_VERBOSE true
   setenv USER_MEM_ARGS "-Xms256m -Xmx1024m -XX:MaxPermSize=512m -XX:CompileThreshold=8000"
   ```

   **bash commands:**
   ```
   export PATH=/path/to/python/bin:$PATH
   export HOME=/path/to/user_dir (Important: If this variable is not set, the FADevConfigure.py script, which is called while creating an Oracle Fusion domain in JDeveloper, will fail.)
   export MW_HOME=/path/to/JDeveloper/installation/directory
   export JAVA_HOME=path/to/installed/64-bit/jdk (Important: A 32-bit JDK is installed into $MW_HOME when you install JDeveloper. If you are running the recommended 64-bit operating system, you will need to install a 64-bit JDK and set the JAVA_HOME variable to it.)
   export PATH=$JAVA_HOME/bin:$PATH
   export JDEV_USER_HOME=/path/to/a/directory (Optional) export FADEV_VERBOSE=true
   export USER_MEM_ARGS="-Xms256m -Xmx1024m -XX:MaxPermSize=512m -XX:CompileThreshold=8000"
   ```

   **Windows command prompt commands:**
   ```
   set HOME=path\to\user_dir (Important: If this variable is not set, the FADevConfigure.py script, that is called while creating an Oracle Fusion domain in JDeveloper, will fail. In Windows 7, HOME typically is c:\users\userid, such as c:\users\herbie.)
   set MW_HOME=path\to\JDeveloper\installation\directory (Important: The path must not contain spaces.)
   set JAVA_HOME=path\to\installed\64-bit\jdk (Important: A 32-bit JDK is installed into $MW_HOME when you install JDeveloper. If you are running the recommended 64-bit operating system, you will need to install a 64-bit JDK and set the JAVA_HOME variable to it. You can download a 64-bit JDK from the Oracle"

set PATH=%JAVA_HOME%\bin;%PATH%
set JDEV_USER_HOME=\path\to\a\directory (Important: The path must not contain spaces.)
set USER_MEM_ARGS=-Xms256m -Xmx1024m -XX:MaxPermSize=512m -XX:CompileThreshold=8000
(Optional) set FADEV_VERBOSE=true

3. Change directory to $MW_HOME/jdeveloper/jdev/bin.

4. Open the jdev.conf file in a text editor and add this line:

   AddVMOption -Djdev.wlst.env.vars=HOME,JDEV_USER_HOME

   JDeveloper checks the environment before running Oracle WebLogic Scripting Tool or starting Oracle WebLogic Server. The domain creation scripts require access to certain environment variables. jdev.wlst.env.vars allows these environment variables to be accessed.

   Note that the HOME variable exists by default in Linux systems but must be set explicitly in Windows systems. See "Windows command prompt commands" in Step 2.

   If necessary, add other options from Table 2-1.

   Save and close the jdev.conf file.

---

### Table 2-1  VMOptions in jdev.conf

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Dide.extension.extra.search.path</td>
<td>/x/y/z</td>
<td>Fully qualified path or paths to the directory or directories where Customization Extension Bundles are located. You probably will need to get this information from an administrator.</td>
</tr>
<tr>
<td>-DURLChooser.forceUseList</td>
<td>true</td>
<td>Optional.</td>
</tr>
<tr>
<td>-DURLChooser.disableCompletionPopup</td>
<td>true</td>
<td>Optional.</td>
</tr>
<tr>
<td>-DUNIX_WEB_BROWSER</td>
<td>/usr/bin/firefox</td>
<td>Location of the browser executable specific to the environment at the customer's site.</td>
</tr>
<tr>
<td>-Dbo.SecondaryADFLibVisible</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>-Ddeployment.jario.writepolicy</td>
<td>recreate</td>
<td></td>
</tr>
<tr>
<td>-Doracle.webcenter.portlet.enableApplicationStriping</td>
<td>true</td>
<td></td>
</tr>
<tr>
<td>-Doracle.webcenter.portlet.dt.excludeExportSet</td>
<td>true</td>
<td></td>
</tr>
</tbody>
</table>
Table 2-1 (Cont.) VMOptions in jdev.conf

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Dadflib.project.open.refresh</td>
<td>false</td>
<td></td>
</tr>
<tr>
<td>-Djdev.wlst.env.vars</td>
<td>HOME, JDEV_US, ER_HOME</td>
<td>These are sanctioned environment variables that will be allowed for use in WebLogic Server Scripting Tool (WLST) scripts. Environment variables that are not in this list will be ignored by the script. You can add variables to this list.</td>
</tr>
<tr>
<td>-XX:MaxPermSize</td>
<td>512M</td>
<td></td>
</tr>
<tr>
<td>-Xmx1024M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-Xms256M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-XX:+DisableExplicitGC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. Start JDeveloper.
   
   **Linux**
   
   jdev &
   
   **Windows**
   
   jdeveloper.exe
   
   If you need more details about the JDeveloper startup, you can use the *verbose* command line argument, such as:
   
   jdev -verbose
   
   When prompted, select the **Default Role**, as shown in Figure 2-1.

**Figure 2-1 Selecting the Default Role**
Because you will need to select a different role later, you should make sure you select the **Always prompt for role selection on startup** option.

The JDeveloper environment can be tailored based on the role you select. The modified environment removes unneeded items from JDeveloper, including menus, preferences, New Gallery, and even individual fields on dialogs. The JDeveloper role you select determines which technologies and options are available to you as you work in JDeveloper.

Table 2-2 provides a brief explanation of the available roles.

<table>
<thead>
<tr>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Role</td>
<td>This role allows you general access to all of JDeveloper’s features without role-specific customizations. The other roles provide important <em>shaping</em> features that are not included in the Default Role. Some of these will be available once the Oracle Fusion Applications Extensions have been added to the IDE.</td>
</tr>
<tr>
<td>Oracle Fusion Applications Administrator Customization</td>
<td>This is the main customization role for Oracle Fusion Applications customers. <strong>Important:</strong> You <em>must</em> use this Role for customizing SOA Composites.</td>
</tr>
<tr>
<td>Oracle Fusion Applications Developer</td>
<td>This is for Oracle Fusion Applications developers to use to build new applications.</td>
</tr>
<tr>
<td>Database Edition</td>
<td>This gives you access to just the core database development tools.</td>
</tr>
<tr>
<td>Java EE Edition</td>
<td>This includes only features for core Java EE development.</td>
</tr>
<tr>
<td>Java Edition</td>
<td>This includes only features for core Java development.</td>
</tr>
<tr>
<td>BPM Process Analyst</td>
<td>Configures the product for a BPM Business Analyst.</td>
</tr>
<tr>
<td>Customization Developer</td>
<td>Configures the product for customizing metadata.</td>
</tr>
<tr>
<td>Oracle Fusion Applications Customization</td>
<td>Configures JDeveloper for customizing metadata for Oracle Fusion Applications developers.</td>
</tr>
</tbody>
</table>

Click **OK**. As JDeveloper loads, the **Migrate User Settings** prompt may display. If it does and you are not sure whether to migrate settings, you should click **No**.

6. Install the **Fusion Apps Development Environment** extension bundle, an all-encompassing JDeveloper bundle that is specific for Oracle Fusion Applications. To install the bundle:

   a. Select **Help > Check for Updates**.

   b. Click **Next** past the Welcome dialog.

   c. If the Proxy Setup dialog displays, enter the applicable information for your situation, as shown in **Figure 2-2**.
Figure 2-2 Completing the Proxy Setup Dialog

Ignore any error messages that might be displayed when you click the Test Proxy button.

d. Click OK. The Source dialog, shown in Figure 2-3, displays.

Figure 2-3 Available Update Centers

Unselect any marked Update Centers.

e. Click Add to display the Update Center dialog, shown in Figure 2-4.
g. Enter a name for the new Update Center, such as Oracle Fusion Applications Update Center.

h. Click **Browse** to locate and select the `fusion_apps_update_center.xml` file, as shown in Figure 2-5. The location of this file is on the Oracle Fusion Applications Companion 11g (11.1.1.7.1) disk in the `fusion_apps_extensions` directory, or on a shared directory provided by the administrator.

![Figure 2-5 Browsing for the Update File](image)

i. Click **Open**.

j. Click **OK**.

k. Select the newly-defined Update Center, as shown in Figure 2-6.

![Figure 2-6 Selecting the New Update Center](image)

Note that only the new Update Center should be selected.
l. Click Next to display the Updates dialog, shown in Figure 2-7. Note that, initially, no updates will be selected.

**Figure 2-7 Selecting the Updates**

![Figure 2-7 Selecting the Updates](image)

m. Select the Fusion Apps Development Environment extension bundle. The other bundles automatically will be selected.

n. Click Next to display the Download dialog and start the download, as shown in Figure 2-8.

**Figure 2-8 Downloading the Updates**

![Figure 2-8 Downloading the Updates](image)

When the download finishes, the Summary dialog, similar to that shown in Figure 2-9, displays automatically.
Click **Finish**. If you are using JDeveloper on a Linux system, the Confirm Exit prompt shown in Figure 2-10 displays.

Click **Yes** to exit JDeveloper.

**Note:** If you are using JDeveloper on Windows the prompt will ask if you want to restart JDeveloper. If you click **Yes**, JDeveloper is automatically restarted.

7. Restart JDeveloper, selecting either the **Oracle Fusion Applications Developer** role or the **Oracle Fusion Applications Administrator Customization** role, as shown in Figure 2-11. The restart results in installing the extension bundles that were downloaded.
If you do not have an administrator-supplied `fusion_apps_wls.properties` file in the default location, you will be prompted to configure WebLogic Server (launch the Oracle Fusion Domain wizard) as shown in Figure 2-12. Click Yes. See How to Use the Oracle Fusion Domain Wizard.

**Note:**

If you do not have access to a database and LDAP server, click No. Integrated WebLogic Server requires access to a valid database and LDAP server.

Note that if you do have an administrator-supplied `fusion_apps_wls.properties` file, the administrator also must supply the entire `o.jdevimpl.rescat2` folder from his `$JDEV_USER_HOME/system11.1.1.xx.yy.zz` folder. See Distributing the `fusion_apps_wls.properties` and `cwallet.sso` Files.

**Figure 2-12  WebLogic Server Not Configured Prompt**

When JDeveloper and Oracle WebLogic Server are properly installed and configured, the environment should have all the components in the correct locations. You should be able to create new, or customize existing, Oracle Fusion applications.

### 2.2.1.6 Use the OWSM_MDS Schema

The administrator has to make sure a database includes the Oracle Web Services Manager_Metadata Service (OWSM_MDS) schema. Typically, and particularly in a test environment and a production environment, this schema is in the Oracle Identity Manager (IDM) database. The administrator probably will not provide the shared environment schema connections because the schema would be locked down. The administrator should install a new MDS schema somewhere and provide that connection.

There are two options for how a developer can use the OWSM_MDS schema.

- The recommended method is for the administrator to use the Oracle Fusion Applications Repository Creation Utility to create the MDS schema, described in Creating the OWSM_MDS Schema, in the database in the shared environment. The schema name must have a prefix of **owsm**. This will create a schema named OWSM_MDS. Developers, then, will not need to do anything else to use the schema.

- The developers have to provide the connect string and other details of the owsm_mds schema available in the central IDM or LDAP while creating the properties file. This requires that an administrator provide the IDM database details. These are the host, port, userid, password, and SID. Figure 2-13 shows how the wizard’s Database dialog will be completed.
2.2.1.7 Distribute the fusion_apps_wls.properties and cwallet.sso Files

Instead of each developer creating the properties file and the credential store using the Oracle Fusion Domain wizard, the administrator can create them once and distribute them to the entire development team. The administrator can use the wizard and enter the property values, which include connect strings to the database and to the Identity Store. These values are captured in the fusion_apps_wls.properties file and the passwords are stored in an encrypted form using the credential store. Both the fusion_apps_wls.properties file and the cwallet.sso file, which is the credential store, are created in the o.jdevimpl.rescat2 sub-folder under the $JDEV_USER_HOME/system11.1.1.7.xx.yy.zz folder. The administrator can distribute the entire o.jdevimpl.rescat2 sub-folder to the development team. The developers can install JDeveloper and install the bundles. The developers then can copy the entire o.jdevimpl.rescat2 sub-folder under their own $JDEV_USER_HOME/system11.1.1.7.xx.yy.zz folder. This way, the administrator can enforce uniformity and the developers will not have to go through the wizard to create the properties file.

Now, if developers need to use their own SOAINFRA or MDS_SOA schemas, they can manually launch the wizard and provide connect strings specifically for those schemas.

Note:

Although the Oracle Fusion Domain wizard includes additional properties for Standalone WebLogic Server, the same fusion_apps_wls.properties and cwallet.sso files are used for both Integrated WebLogic Server and Standalone WebLogic Server creation and configuration.

2.2.2 How to Use the Oracle Fusion Domain Wizard

The wizard helps you to create and update a fusion_apps_wls.properties file and a cwallet.sso file that are used to set up the Oracle WebLogic Server domain for Oracle Fusion Applications development. The wizard incorporates two main paths: one for configuring an Integrated WebLogic Server domain (in which JDeveloper manages the server) and one for setting up a remote Standalone WebLogic Server domain.

In the case of Integrated WebLogic Server, completion of the wizard will create the domain. For Standalone WebLogic Server, you will have to create the domain from the command line, using a Python script and the fusion_apps_wls.properties and cwallet.sso files that were populated using the wizard.

The wizard can be run multiple times to change properties in the fusion_apps_wls.properties file. If certain critical properties are changed, the
domain will have to be re-created. This will be done automatically for the Integrated
WebLogic Server domain, but will be a manual step for a Standalone WebLogic Server
domain.

The wizard can be launched automatically or manually. It will be launched
automatically under either of these conditions:

- The \texttt{fusion\_apps\_wls.properties} file is not found when JDeveloper starts.
The name of the file defaults to \texttt{fusion\_apps\_wls.properties}, and the
location defaults to the \texttt{system1.1.xx.yy.zz/o.jdevimpl.rescat2}
directory.

- The \texttt{fusion\_apps\_wls.properties} file is not found when you select to \textbf{Run}
an application from JDeveloper, or you select to \textbf{Start Server Instance} from
JDeveloper.

To start the wizard manually from within JDeveloper:

- Select \textbf{View} > \textbf{Application Server Navigator}.

- Expand the \textbf{Application Servers} node.

- Right-click \textbf{IntegratedWebLogicServer} and select the \textbf{Configure Fusion Domain}
option, as shown in \textbf{Figure 2-14}.

\textbf{Figure 2-14} \hspace{1cm} \textbf{Manually Starting the Oracle Fusion Domain Wizard}

The properties that can be captured in the wizard are shown in Table 2-3. The
properties are defined under section headers that are surrounded by [square brackets],
for example:

\begin{verbatim}
[domain]
domainType=adminall
\end{verbatim}
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Standalone/Integrated</th>
<th>Required</th>
<th>Default</th>
<th>Comments</th>
</tr>
</thead>
</table>
| domainType    | Standalone             | Yes      | standalone | Valid values are:  
|               |                        |          |          | **adfess**: Admin server only for Oracle ADF and ESS  
|               |                        |          |          | **adminsoa**: Admin server (EM) and SOA managed server  
|               |                        |          |          | **adminall**: Combination of standalone and adminsoa (Oracle ADF, ESS, SOA)  
|               |                        |          |          | **adminessadf**: Admin server for Oracle ADF and ESS managed server  
|               |                        |          |          | **standalone**: Default setting. This is an admin server only; not a managed server. |
| domainName    | Standalone             | Yes      | fusion_domain | This can be any name. It is used only for Standalone Weblogic Server. |
| domainDir     | Standalone             | Yes      |          | The location in the file system in which the domain will be created. To avoid possible errors during script execution, the directory name should not include spaces.  
|               |                        |          |          | If it is not specified, the domain will be created in the default location which is $MW_HOME/user_projects/<domain_name>. |
| installerLocation | Standalone          | Yes      |          | Location of the Oracle Fusion Applications installer files. These usually are located on a central server, rather than on each developer's system. Ask your administrator for the location. |
| listenPort    | Standalone             | Yes      | Default based on Standalone or Integrated. Default is 7011 for Standalone WebLogic Server. |
| soaPort       | Standalone             | Yes, only when domainType is adminsoa/adminall. |          | Needed for adminsoa and adminall.  
<p>|               |                        |          |          | As this is on their own machines, developers choose the values. |
| essPort       | Standalone             | Yes, only when domainType is adminess. |          | As this is on their own machines, developers choose the values. |
| wlName        | Both                   | Yes      | weblogic |          |</p>
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Standalone/Integrated</th>
<th>Required</th>
<th>Default</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>wlPassword</td>
<td>Both</td>
<td>Yes</td>
<td>NA</td>
<td>Users must supply the password by using the wizard.</td>
</tr>
<tr>
<td>ldapHost</td>
<td>Both</td>
<td>Yes</td>
<td></td>
<td>This is a string value similar to: ldaphostname.yourcompany.com.</td>
</tr>
<tr>
<td>ldapPort</td>
<td>Both</td>
<td>Yes</td>
<td></td>
<td>Example: 3060</td>
</tr>
<tr>
<td>ldapUser</td>
<td>Both</td>
<td>Yes</td>
<td></td>
<td>Example: cn=wlsproxyuser</td>
</tr>
<tr>
<td>ldapPass</td>
<td>Both</td>
<td>Yes</td>
<td></td>
<td>Example: welcome1 Important: In the UI, the password will display as the normal ******* mask. The password is not saved in the fusion_apps_wls.properties file. It is encrypted and saved in the cwallet.sso file maintained in the system11.1.1.xx.yy.zz/o.jdevimpl.rescat2/directory.</td>
</tr>
<tr>
<td>ldapUserDN</td>
<td>Both</td>
<td>Yes</td>
<td></td>
<td>Example: cn=users,dc=us,dc=yourcompany,dc=com</td>
</tr>
<tr>
<td>ldapGroupDN</td>
<td>Both</td>
<td>Yes</td>
<td></td>
<td>Example: cn=groups,dc=us,dc=yourcompany,dc=com</td>
</tr>
<tr>
<td>ldapSSLEnabled</td>
<td>Both</td>
<td>No</td>
<td>false</td>
<td>true or false</td>
</tr>
<tr>
<td>Property Name</td>
<td>Standalone/Integrated</td>
<td>Required</td>
<td>Default</td>
<td>Comments</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------</td>
<td>----------</td>
<td>---------</td>
<td>----------</td>
</tr>
</tbody>
</table>
| opssHost      | Standalone            | No       |         | Optional separate OPSS store. This is a string value similar to: opsshost\name\yourcompany.com. For Integrated WebLogic Server, the DefaultDomain uses the XML-based Policy Store (such as system-jazn-data.xml). For Standalone WebLogic Server, if undefined, the standalone domain defaults to the XML-based Policy Store (such as system-jazn-data.xml). For information and directions on how to migrate the OPSS policy store using the migrateSecurityStore.py OPSS script, see:  
• "Migrating the OPSS Security Store" and "Migrating with the Script migrateSecurityStore" in the Oracle Fusion Middleware Applications Security Guide.  
• "Extracting Data from an OID Store to a File" in the Administrator’s Guide. |
| opssPort      | Standalone            | No       |         | Example: 3061 |
| opssUser      | Standalone            | No       |         | Example: cn=wlsproxyuser |
| opssPass      | Standalone            | No       |         | Important: In the UI, the password will display as the standard "*****" mask. The password is not saved in the fusion\apps_wls.properties file. It is encrypted and saved in the cwallet.sso file maintained in the system11.1.1.xx.yy.zz/o.jdevimpl.rescat2/ directory. |
| opssSSLEnabled| Standalone            | No       | false   | true or false |
| jpsRootContext| Standalone            | No       |         | Text field  
Example: cn=FADevPolicies |
| biHostPort    | Standalone            | No       |         | Oracle Business Intelligence will only be supported in the Standalone WebLogic Server environment. Specify a port to point to the BI server. This generates a BIP configuration file that gets added to the domain home. The template then adds the location of the configuration file as a system property to be set when Oracle WebLogic Server is started. |
Table 2-3  (Cont.) Properties to be Captured in the Oracle Fusion Domain Wizard

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Standalone/ Integrated</th>
<th>Required</th>
<th>Default</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>familyName</td>
<td>Both</td>
<td>Yes</td>
<td></td>
<td>Family names are: COMMON, IC, HCM, FIN, PRC, PRJ, SCM, and CRM.</td>
</tr>
<tr>
<td>fusionDb - connect string</td>
<td>Both</td>
<td>Yes</td>
<td></td>
<td>JDBC connect string. Note: This is split into sub-fields:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• fusionDbUser</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• fusionDbPassword</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• fusionDbHost</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• fusionDbPort</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• fusionDbSid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>This is applicable to these schemas:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• activityDb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• apmDb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• AppMasterDb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• essMdsDb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• fusionAq</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• fusionEdn</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• fusionMds</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• oraessDb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• orasdpmDb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• owsmMdsDb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• portletDb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• soadatasrcDb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• soaMdsDb</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• wcDb</td>
</tr>
</tbody>
</table>

activityDbUser - connect string | Both | No | You can connect to the fusionDB database as fusion_activities. Your system administrator will provide the connection details and schema password to be used. |

apmDbUser - connect string | Both | No | You can connect to the fusionDB database as fusion_apm. Your system administrator will provide the connection details and schema password to be used. |

AppMasterDbUser - connect string | Both | No | You can connect to the fusionDB database as fusion_runtime. Your system administrator will provide the connection details and schema password to be used. |

essMdsDbUser - connect string | Both | No | You can connect to the fusionDB database as fusion_mds_ess. Your system administrator will provide the connection details and schema password to be used. |
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Standalone/Integrated</th>
<th>Required</th>
<th>Default</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>fusionAqUser</td>
<td>Both</td>
<td>Yes</td>
<td>No</td>
<td>You can connect to the fusionDB database as fusion_aq. Your system administrator will provide the connection details and schema password to be used.</td>
</tr>
<tr>
<td>- connect string</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fusionEdnUser</td>
<td>Both</td>
<td>Yes</td>
<td>No</td>
<td>Note: CRM will be replaced with the familyName that is passed in.</td>
</tr>
<tr>
<td>- connect string</td>
<td></td>
<td></td>
<td></td>
<td>You can connect to the fusionDB database as crm_fusion_soainfra. Your system administrator will provide the connection details and schema password to be used.</td>
</tr>
<tr>
<td>fusionMdsUser</td>
<td>Both</td>
<td>Yes</td>
<td>No</td>
<td>You can connect to the fusionDB database as fusion_mds. Your system administrator will provide the connection details and schema password to be used.</td>
</tr>
<tr>
<td>- connect string</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>oraEssDbUser</td>
<td>Both</td>
<td>No</td>
<td>No</td>
<td>You can connect to the fusionDB database as fusion_ora_ess. Your system administrator will provide the connection details and schema password to be used.</td>
</tr>
<tr>
<td>- connect string</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>orasdpmDbUser</td>
<td>Both</td>
<td>No</td>
<td>No</td>
<td>You can connect to the fusionDB database as fusion_orasdpm. Your system administrator will provide the connection details and schema password to be used.</td>
</tr>
<tr>
<td>- connect string</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>owsmMdsDbUser</td>
<td>Both</td>
<td>No</td>
<td>No</td>
<td>You can connect to the fusionDB database as owsm_mds. Your system administrator will provide the connection details and schema password to be used.</td>
</tr>
<tr>
<td>- connect string</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>portletDbUser</td>
<td>Both</td>
<td>No</td>
<td>No</td>
<td>You can connect to the fusionDB database as fusion_portlet. Your system administrator will provide the connection details and schema password to be used.</td>
</tr>
<tr>
<td>- connect string</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>soadatasrcDbUser</td>
<td>Both</td>
<td>No</td>
<td>No</td>
<td>Family name used in prefix.</td>
</tr>
<tr>
<td>- connect string</td>
<td></td>
<td></td>
<td></td>
<td>You can connect to the fusionDB database as crm_fusion_soainfra. Your system administrator will provide the connection details and schema password to be used.</td>
</tr>
</tbody>
</table>
Table 2-3  (Cont.) Properties to be Captured in the Oracle Fusion Domain Wizard

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Standalone/Integrated</th>
<th>Required</th>
<th>Default</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>soaMdsDbUser - connect string</td>
<td>Both</td>
<td>No</td>
<td></td>
<td>Family name used in prefix. You can connect to the fusionDB database as crm_fusion_mds_soa. Your system administrator will provide the connection details and schema password to be used.</td>
</tr>
<tr>
<td>wcDbUser - connect string</td>
<td>Both</td>
<td>No</td>
<td></td>
<td>You can connect to the fusionDB database as fusion_webcenter. Your system administrator will provide the connection details and schema password to be used.</td>
</tr>
</tbody>
</table>

2.2.2.1 Creating the Properties File for Default Integrated Server

Note:
The wizard requires considerable information about the network and various servers, such as LDAP and database. In normal situations, the administrator will disseminate his entire o.jdevimpl.rescat2 folder from his $JDEV_USER_HOME/system11.1.1.xx.yy.zz folder to developers who will copy the folder into the correct directory. See Distributing the fusion_apps_wls.properties and cwallet.sso Files.

If the fusion_apps_wls.properties file is not found when you start JDeveloper, the prompt shown in Figure 2-12 displays. You also can start the wizard manually. Click Yes to launch the wizard and display the Usage page, as shown in Figure 2-15.
Figure 2-15  Selecting Domain Usage

- **Default Integrated Server**
  Select this option, the default, to configure and create a server that will be controlled by JDeveloper. This is the normal choice for development work. When the wizard finishes, an Integrated WebLogic Server domain will be created and can be used to run and test your applications.

- **Standalone Server**
  Selecting this option only creates or updates the fusion_apps_wls.properties and cwallet.sso files. See Completing the Oracle Fusion Domain Wizard for Standalone Server for the dialogs that are specific to creating the fusion_apps_wls.properties file for a Standalone WebLogic Server domain. Creating a Standalone WebLogic Server domain must be done from the command line.

- **Messages**
  A message is displayed in this field if any errors occur in the definition. These errors must be corrected before you continue.

Further wizard pages depend on the selected Usage. The flow for the **Default Integrated Server** selection is covered first. The flow for the **Standalone Server** selection is covered in Completing the Oracle Fusion Domain Wizard for Standalone Server.

When you select the **Default Integrated Server** Usage option and click **Next**, the Domain dialog, shown in Figure 2-16, displays.
If the `fusion_apps_wls.properties` file already exists and is in place, the fields will show the values that are in the file.

- **WebLogic User Name**
  This value defaults to `weblogic`. Change it if necessary.

- **WebLogic Password / Confirm Password**
  The password requires at least one numeral. There is no default, and you must supply the password. Note that the password is not stored in the `fusion_apps_wls.properties` file. It is encrypted and stored in the `cwallet.sso` file.

- **Fusion Family Name**
  This name is used during Java Message Service queue creation; and as the SOA cluster name when creating an instance of Standalone Weblogic Server. The value should match the product you are customizing.

- **Messages**
  A message is displayed in this field if any errors occur in the definition. These errors must be corrected before you continue.

Click **Next** to display the Database dialog, shown in **Figure 2-17**.
If the `fusion_apps_wls.properties` file already exists and is in place, the fields will show the values that are in the file.

**Note:** If you do not have access to a database, click **Cancel**.
Note:

- Values entered in the wizard are not validated, so the connection details must be validated outside the wizard.

- If a credential that is listed in the Database dialog does not have the same password as the corresponding schema name itself, you must provide the password using the fields to the right of the credential list.

- If the credential that is listed in the Database dialog has to be mapped to a schema other than the default schema, you should provide the appropriate schema name and password using the fields to the right of the credential list. For example, the OWSM MDS credential is mapped to the owsm_mds schema by default. But, if it has to be mapped to the hcm_fusion_mds_soa schema, you can choose the OWSM MDS credential from the list and then enter hcm_fusion_mds_soa in the Username field and specify the password in the Password field on the right. Again, the Connect String field can be left blank if the schema is in the same database.

- If each of the credentials in the Database dialog have the same password as the corresponding schema name, you only need to enter values in the Fusion Database field in the host:port:SID format.

- Make sure you have entered valid database connection details or your WebLogic Servers will not start. The ApplicationDB data source that is configured using these values is configured to support global transaction type Logging Last Resource. If the database is not available when you try to start WebLogic Server, it will fail with a message similar to:

  Server failed. Reason: JTAExceptions:119002A logging last resource failed during initialization. The server cannot boot unless all configured logging last resources (LLRs) initialize

- Fusion Database User
  This schema name comes from the database installation. fusion_runtime is a recommended standard name.

- Fusion Database Password
  Enter the password. You probably will need to get this from an administrator if the cwallet.sso file was not provided to you. (Passwords are encrypted and stored in that file.)

- Fusion Database
  Enter the host, port, and the SID information using a colon (:) delimiter, such as a.your.company.com:1234:xyzzyon. You probably will need to get this from an administrator if the fusion_apps_wls.properties file was not prepared for you.

- Credential Type List
  Several credentials are supplied with Oracle Fusion Applications and are included in the fusion_apps_wls.properties file. When you click a credential, the three fields to the right will display the default values. The Password field will
remain blank because any passwords are encrypted and stored in the
cwallet.sso file.

If OWSM_MDS is selected, and the administrator has chosen to open up the IDM
database in which the schema already exists, you will need to enter all the
necessary information in this dialog. However, if the administrator has created the
OWSM_MDS schema in the transaction database, you may not need to enter any
data here. For more information about the owsm_mds schema, see Creating the
OWSM_MDS Schema.

You can change the default values of almost all the credentials, if necessary.

- **User Name**
  
  This field corresponds to the Fusion Database User field.

- **Password**
  
  This field corresponds to the Fusion Database Password field.

- **Connect String**
  
  This field corresponds to the Fusion Database field.

- **Messages**

  A message is displayed in this field if any errors occur in the definition. These
  errors must be corrected before you continue.

Click **Next** to display the Security dialog, shown in Figure 2-18.

**Figure 2-18 Configuring Security for Integrated Domain Server**

If the fusion_apps_wls.properties file already exists and is in place, the fields
will show the values that are in the file.
Note: If you do not have access to an LDAP server, click Cancel.

- **LDAP Server**
  This field cannot be edited directly. Click the edit icon to display the Edit LDAP Server dialog shown in Figure 2-19.

*Figure 2-19 Editing the LDAP Server*

- **Host**
  Enter the name of your LDAP host, such as `ldap_server.your_company.com`.

- **Port**
  Enter the port number, such as `1066`.

- **Principal**
  This is the internal LDAP user name by which you connect to LDAP, such as `cn=wlsproxyuser`.

- **Password**
  Enter the password used by the Principal. The password will be encrypted and stored in the `cwallet.sso` file, and not in the `fusion_apps_wls.properties` file.

- **SSL Enabled**
  This value defaults to LDAP (not checked). Select this check box if you want to use LDAPS.

- **User Base DN**
  Enter the User DN based your LDAP. A sample User DN resembles `cn=users,dc=us,dc=your_company,dc=com`.
  The DN (Distinguished Name) is the LDAP attribute that uniquely defines an object. Each DN must have a unique name and location.
  The components include `cn=common name, ou=organizational unit, and dc=domain content`. DC often is listed with two entries, `dc=cp` and `dc=com`.

- **Group Base DN**
  Enter the Group DN based your LDAP. A sample Group DN resembles `cn=groups,dc=us,dc=your_company,dc=com`.
• **Messages**

A message is displayed in this field if any errors occur in the definition. These errors must be corrected before you continue.

Click **Next** to display the Finish dialog shown in **Figure 2-20**.

**Figure 2-20  Finishing the Integrated Domain**

- **Click Finish to save the properties in the following file**

  This value cannot be edited. The field simply shows the name of the `fusion_apps_wls.properties` file and the directory in which it will be created or updated.

- **Create the Domain Now**

  This defaults to Yes (checked). When selected and you click **Finish**, the Integrated WebLogic Server domain will be created so you can test your applications by selecting one of the JDeveloper **Run** options.

  **Note:**

  Creating the domain involves a great deal of background work to correctly set up the environment. This process can take several minutes.

2.2.2.2 **Complete the Oracle Fusion Domain Wizard for Standalone Server**

When you select the Standalone Server Usage option and click **Next**, the Domain dialog, shown in **Figure 2-21**, displays.
Figure 2-21 Configure the Standalone Domain

If the fusion_apps_wls.properties file already exists and is in place, the fields will show the values that are in the file.

Note:

If an administrator has not created the fusion_apps_wls.properties file for you, with this information, you will need to get most of this information from an administrator.

- **Domain Type**
  This is the type of domain you wish to create. It can be:
  
  - **adfess**: Configured for Oracle ADF and Oracle Enterprise Scheduler technologies.
  
  - **adminsoa**: Creates an Oracle SOA Suite domain with an AdminServer with Oracle Enterprise Manager Fusion Middleware Control deployed, and a managed server, named soa_server1, where SOA composites are deployed and run.
  
  - **adminall**: A combination of standalone and Oracle SOA Suite, AdminServer for Oracle ADF/Oracle Enterprise Scheduler and managed server soa_server1 for Oracle SOA Suite.
  
  - **adminessadf**: Admin server for Oracle ADF and Oracle Enterprise Scheduler Service (ESS) managed server.

- **Domain Name**
The name of your domain. If you have more than one domain, you need to change this value and the domainDir value.

Follow these steps if you are using a Linux operating system:

- Create a new fusion_apps_wls.properties file with domainName=Domain1 and domainDir=$MW_HOME/user_projects/Domain1.

- When the properties file has been created, copy fusion_apps_wls.properties to fusion_apps_wls_Domain1.properties in the $JDEV_USER_HOME/system11.1.1.*/o.jdevimpl.rescat2 directory.

- Run FADevCreateDomain.py -p $JDEV_USER_HOME/system11.1.1.*/o.jdevimpl.rescat2/fusion_apps_wls_Domain1.properties to create Domain1.

- Edit fusion_apps_wls.properties and change domainName=Domain2 and domainDir=$MW_HOME/user_projects/Domain2.

- Copy fusion_apps_wls.properties to fusion_apps_wls_Domain2.properties in the $JDEV_USER_HOME/system11.1.1.*/o.jdevimpl.rescat2 directory.

- Run FADevCreateDomain.py -p $JDEV_USER_HOME/system11.1.1.*/o.jdevimpl.rescat2/fusion_apps_wls_Domain2.properties to create Domain2.

You now have two properties files in the o.jdevimpl.rescat2 folder and two domains.

Follow these steps if you are using a Windows operating system:

- Create a new fusion_apps_wls.properties file with domainName=Domain1 and domainDir=%MW_HOME%\user_projects\Domain1.

- cd %JDEV_USER_HOME%\system11.1.1.*\o.jdevimpl.rescat2

- copy fusion_apps_wls.properties fusion_apps_wls_Domain1.properties

- Run FADevCreateDomain.py -p %JDEV_USER_HOME%\system11.1.1.*\o.jdevimpl.rescat2\fusion_apps_wls_Domain1.properties to create Domain1.

- Edit fusion_apps_wls.properties and change domainName=Domain2 and domainDir=%MW_HOME%\user_projects\Domain2.

- cd %JDEV_USER_HOME%\system11.1.1.*\o.jdevimpl.rescat2

- copy fusion_apps_wls.properties fusion_apps_wls_Domain2.properties

- Run FADevCreateDomain.py -p %JDEV_USER_HOME%\system11.1.1.*\o.jdevimpl.rescat2\fusion_apps_wls_Domain2.properties to create Domain2.
You now have two properties files in the o.jdevimpl.rescat2 folder and two domains.

- **Installer Location**
  This is the location of the Oracle Fusion Applications installer files, usually on a central server.

- **Domain Location**
  The location in the file system in which the domain will be created.
  If it is not specified, it will be created in the default location, which is $MW_HOME/user_projects/<domain_name>.
  This can be changed by setting the domainDir property in the fusion_apps_wls.properties file. If you have more than one domain, you need to change this value. See the description of domainName.

- **Listen Port**
  Listen Port is the adminserver listen-port you want to use.

- **SOA Port**
  This field, which displays only if the Domain Type is adminall or adminsoa, is the port number for your Oracle SOA Suite managed server (soa_server1) if you are using Oracle SOA Suite. If you are not using Oracle SOA Suite, you can leave this blank.

- **BI Host Port**
  This is the host:port where the BI Publisher server is running. The format for the value for this field is hostname:port, such as my.domain.com:9999. You may need to get this value from your administrator.

- **WebLogic User Name**
  This value defaults to weblogic. Change it if necessary.

- **WebLogic Password / Confirm Password**
  The password requires at least one numeral. There is no default, and you must supply the password. Note that the password is not stored in the fusion_apps_wls.properties file. It is encrypted and stored in the cwallet.sso file.

- **Fusion Family Name**
  This name is used during Java Message Service queue creation; and as the SOA cluster name when creating an instance of Standalone Weblogic Server. The value should match the product you are customizing.

- **Messages**
  A message is displayed in this field if any errors occur in the definition. These errors must be corrected before you continue.

Click **Next** to display the Database dialog, shown in Figure 2-22
If the `fusion_apps_wls.properties` file already exists and is in place, the fields will show the values that are in the file.

**Note:** If you do not have access to a database, click **Cancel**.

**Note:**

- Values entered in the wizard are *not* validated, so the connection details *must* be validated outside the wizard.

- If a credential that is listed in the Database dialog does not have the same password as the corresponding schema name itself, you *must* provide the password using the fields to the right of the credential list.

- If the credential that is listed in the Database dialog has to be mapped to a schema other than the default schema, you should provide the appropriate schema name and password using the fields to the right of the credential list. For example, the OWSM MDS credential is mapped to the `owsm_mds` schema by default. But, if it has to be mapped to the `hcm_fusion_mds_soa` schema, you can choose the OWSM MDS credential from the list and then enter `hcm_fusion_mds_soa` in the Username field and specify the password in the Password field on the right. Again, the Connect String field can be left blank if the schema is in the same database.

- If each of the credentials in the Database dialog have the same password as the corresponding schema name, you only need to enter values in the Fusion Database field in the `host:port:SID` format.
• **Fusion Database User**
  This schema name comes from the database installation. A recommended standard name is `fusion_runtime`.

• **Fusion Database Password**
  Enter the password. You probably will need to get this from an administrator if the `cwallet.sso` file was not provided to you. (Passwords are encrypted and stored in that file.)

• **Fusion Database**
  Enter the host, port, and the SID information using a colon (:) delimiter, such as `a.your.company.com:1234:xyzzyon`. You probably will need to get this from an administrator if the `fusion_apps_wls.properties` file was not prepared for you.

• **Credential Type List**
  Several credentials are supplied with Oracle Fusion Applications and are included in the `fusion_apps_wls.properties` file. When you click a credential, the three fields to the right will display the default values. The Password field will remain blank because any passwords are encrypted and stored in the `cwallet.sso` file.

  If OWSM_MDS is selected, and the administrator has chosen to open up the IDM database in which the schema already exists, you will need to enter all the necessary information in this dialog. However, if the administrator has created the OWSM_MDS schema in the transaction database, you may not need to enter any data here. For more information about the `owsm_mds` schema, see `Creating the OWSM_MDS Schema`.

  You can change the default values of almost all the credentials, if necessary.

  – **User Name**
    This field corresponds to the Fusion Database User field.

  – **Password**
    This field corresponds to the Fusion Database Password field.

  – **Connect String**
    This field corresponds to the Fusion Database field.

• **Messages**
  A message is displayed in this field if any errors occur in the definition. These errors must be corrected before you continue.

Click **Next** to display the Security dialog shown in Figure 2-23.
Figure 2-23 Configuring Security for Standalone Server

If the `fusion_apps_wls.properties` file already exists and is in place, the fields will show the values that are in the file.

**Note**: If you do not have access to an LDAP server, click **Cancel**.

- **LDAP Server**
  
  This field cannot be edited directly. Click the edit icon to display the Edit LDAP Server dialog shown in Figure 2-24.

Figure 2-24 Editing the LDAP Server

- **Host**
  
  Enter the name of your LDAP host, such as `ldap_server.your_company.com`.

- **Port**
Enter the port number, such as 1066.

- **Principal**
  This is the internal LDAP user name by which you connect to LDAP, such as cn=wlsproxyuser.

- **Password**
  Enter the password used by the Principal. The password will be encrypted and stored in the cwallet.sso file, and not in the fusion_apps_wls.properties file.

- **SSL Enabled**
  This value defaults to LDAP (not checked). Select this check box if you want to use LDAPS.

- **User Base DN**
  Enter the User DN based your LDAP. A sample User DN resembles cn=users,dc=us,dc=your_company,dc=com.
  
  The DN (Distinguished Name) is the LDAP attribute that uniquely defines an object. Each DN must have a unique name and location.
  
  The components include cn=common name, ou=organizational unit, and dc=domain content. DC often is listed with two entries, dc=cp and dc=com.

- **Group Base DN**
  Enter the Group DN based your LDAP. A sample Group DN resembles cn=groups,dc=us,dc=your_company,dc=com.

- **OPSS Server**
  Defining the Oracle Platform Security Services (OPSS) Server is optional. Define this if its policy store is in a different location than your LDAP policy store. If OPSS-related properties are not specified, the domain is configured to use the XML-based Policy Store, such as system-jazn-data.xml.
  
  For information and directions on how to migrate the OPSS policy store using the migrateSecurityStore.py OPSS script, see:
  
  
  - "Extracting Data from an OID Store to a File" in the Administrator’s Guide.
– **Host**
  Enter the name of your LDAP host, such as `ldap_server.your_company.com`.

– **Port**
  Enter the port number, such as `1066`.

– **Principal**
  This is the internal LDAP user name by which you connect to LDAP, such as `cn=wlsproxyuser1`.

– **Password**
  Enter the password used by the Principal. The password will be encrypted and stored in the `cwallet.sso` file, and not in the `fusion_apps_wls.properties` file.

– **SSL Enabled**
  This value defaults to not enabled. Select this check box if you want to enable SSL.

– **JPS Root Context**
  Enter the JPS Root Distinguished Name, which is the top-level (outermost) node that contains OPSS data in an LDAP directory, such as `cn=FAPolicies`.

Click **Next** to display the Finish dialog shown in **Figure 2-26**.

**Figure 2-26**  **Finishing the Standalone Domain Configuration**

- Click **Finish** to save the properties in the following file.
This value cannot be edited. The field simply shows the name of the fusion_apps_wls.properties file and the directory in which it will be created or updated.

- To create the domain ...

This value cannot be edited. The field simply shows the directory in which the script file will be created, and the name of the script file, FADevCreateDomain.py, you will need to run at the command line. See Creating a Domain for Standalone WebLogic Server.

- Messages

A message is displayed in this field if any errors occur in the definition. These errors must be corrected before you continue.

2.2.3 How to Start Integrated WebLogic Server

You can start Integrated WebLogic Server before running or deploying any applications. If you run an application without having already started Oracle WebLogic Server, JDeveloper will start it.

2.2.3.1 Start Integrated WebLogic Server

There are two ways to start Integrated WebLogic Server.

- Right-click and run a page from a project. If the server is not running, it will be started.
- From the JDeveloper main menu, select Run > Start Server Instance, as shown in Figure 2-27.

Figure 2-27  Start Server Using Start Server Instance

2.2.3.2 Stop Integrated WebLogic Server and the Application

To stop Integrated WebLogic Server or the application from either the Integrated Server window or from the JDeveloper menu bar, click the red stop button and select
either the IntegratedWebLogicServer or the connection option, as shown in Figure 2-28.

Figure 2-28 Stopping Integrated WebLogic Server or the Application

If you select the connection option, the application will be undeployed and the server will remain running.

If you select the IntegratedWebLogicServer option, the deployed application will be undeployed and the server shut down. Wait for the application to be undeployed and the server to stop.

If the shutdown of Integrated WebLogic Server did not respond or shut down the server, click the red shutdown button again to kill the process, as shown in Figure 2-29.

Figure 2-29 Forcing Shutdown of Integrated WebLogic Server

If you still do not see the Process Exited message when you terminate Integrated WebLogic Server, you will have to manually kill the process.

To manually kill the process from Linux, follow these steps:

1. From a terminal window, execute this command.
   
   ```bash
   /usr/sbin/lsof -i -P | grep 7101
   ```

   **Note:**

   7101 is the default Integrated WebLogic Server port. It may be different.

2. Kill the process. For instance, if the process is 15846, you would execute this command:
   
   ```bash
   kill -9 15846
   ```

To manually kill the process from Windows, follow these steps:

1. Start the Windows Task Manager.
2. Click the Processes tab if it is not already active.
3. Click the Image Name column heading to place the processes in an alphabetical order.
4. Locate and select the java.exe process.
2.2.3.3 Manage Integrated WebLogic Server

The WebLogic Server Console is deployed by default and can be accessed to manage Integrated WebLogic Server. To access the WebLogic Server Console, enter the following URL in your web browser: http://<hostname.domainname>:<port>/console, such as http://localhost:7101/console.

2.3 Improve Shared and Personal Environments Performance

This section presents three situations whose security or performance can be improved:

- Creating a local OWSM_MDS Schema
- Using alternate database schemas
- Running Standalone WebLogic Server from the personal environment

2.3.1 How to Create the OWSM_MDS Schema

This step is optional and depends on how the environment is set up and the type of security that is required.

The mds-owsm datasource must be correctly configured to point to an MDS schema. In the production environment, this schema would be in the LDAP infrastructure database.

To configure the datasource, you need to know the schema password. The recommendation is for the administrator to create a new MDS schema and give developers the password for it.
2.3.1.1 Creating the OWSM_MDS Schema

This section provides detailed snapshots on how to create the OWSM_MDS schema using the Oracle Fusion Applications Repository Creation Utility.

In a production environment, the OWSM_MDS schema is in the IDM database. The IDM is typically locked down so information such as schema passwords are not handed out. But, to configure the domain, the schema and the password are required to set up the mds-owsm datasource. The starter transaction database does not contain the OWSM_MDS schema. This is correct and the base template to create the starter transaction database should not be changed to include it. To avoid widely disseminating the schema password of the IDM database, an extra MDS should be added to the development’s transaction database using the Oracle Fusion Applications Repository Creation Utility with a prefix of OWSM.

You do not have to do anything else apart from adding this schema. The schema will be correctly populated when your domain starts, if it does not already contain the correct data.

There may be several Oracle Fusion Applications Repository Creation Utilities available. To provision Oracle Fusion Applications, you will have created an installer repository. In this repository, you will see the following:

```
installers/
  fmw_rcu/
    linux/
      rcuHome.zip
    windows/
      rcuHome.zip
  biapps_rcu/
  fmw_rcu/
```

Copy the appropriate ZIP file to your system from the installers/fmw_rcu directory.

When you get the ZIP file to your machine, follow these steps:

**Linux system**

```
% mkdir fa_rcu
% cd fa_rcu
% cp /from_zip_file_location/rcuHome.zip .
% unzip rcuHome.zip
% cd bin
% ./rcu
```

**Windows system**

```
md fa_rcu
cd fa_rcu
copy \from_zip_file_location\rcuHome.zip .
unzip rcuHome.zip
\path_to_rcu_utility\rcu
```

The Oracle Fusion Applications Repository Creation Utility starts and displays the Create Repository dialog, shown in Figure 2-32.
Select **Create** and click **Next** to display the Database Connection Details dialog, shown in Figure 2-33.

**Figure 2-33  Creating the Database Connection**
Enter your database connection details and click **Next** to display the Checking Global Prerequisites dialog, shown in **Figure 2-34**.

![Figure 2-34 Checking Global Prerequisites](image)

Click **OK** to display the Select Components dialog, shown in **Figure 2-35**.

![Figure 2-35 Selecting Components](image)

Select **Create a new Prefix** and enter OWSM as the name.

Expand **AS Common Schemas** and select Metadata Services.

Click **Next** to display the Checking Component Prerequisites dialog, shown in **Figure 2-36**.
Click **OK** to display the Schema Passwords dialog, shown in **Figure 2-37**.

Select **Use same passwords for all schemas** and enter the password for the OWSM_MDS schema in the **Password** and **Confirm Password** fields.

Click **Next** to display the Map Tablespaces dialog, shown in **Figure 2-38**.
In the **Default Tablespace** field, select FUSION_TS_TOOLS.

In the **Temp Tablespace** field, select either FUSION_TEMP or TEMP3.

Click **Next** to display the Validating and Creating Tablespaces dialog, shown in Figure 2-39.

**Figure 2-39  Validating Created Tablespaces**

Click **OK** to display the Summary dialog, shown in Figure 2-40.
Verify the information and click **Create**.

When the operation has completed successfully, the Completion Summary dialog, shown in Figure 2-41, displays.
The OWSM_MDS schema now can be used to configure the mds_owsm datasource in the domain.

2.3.2 How to Use Alternate Database Schemas

The main reason to use an alternate database is to improve performance. For instance, if the main database is remote, you can improve performance by installing the dehydration store, EDN, MDS and OrasDPM on your local machine.

To use an alternate database schema, follow these steps.

1. Create the required database schemas, as shown in Example 2-2.

   ![Image of Completion Summary]

   Click Close.

   The OWSM_MDS schema now can be used to configure the mds_owsm datasource in the domain.

   **Note:**
   These steps need the number of processes in the database to be set to at least 200. If needed, log in as sysdba, run this command, and restart the database.

   ```sql
   alter system set processes=200 scope=SPFILE;
   ```

2. Drop the Repository, as shown in Example 2-3. Enter the SYS password when prompted.

   If the -silent switch is omitted, a wizard will be launched. It will ask you to enter the same values as shown in Example 2-2.

3. Recreate the Repository, as shown in Example 2-4. Enter the SYS password when prompted.
Note:
You will need to supply passwords for the different users. You will have to remember all the passwords because you will need them when you configure the DataSources.

Example 2-2  Creating database schemas
Follow these steps if you are using a Linux operating system:

```bash
cd $RCU_SHIPHOMELOC/bin
DB_HOST=localhost
DB_PORT=1521
DB_SID=XE
CONNECT_STRING=$DB_HOST:$DB_PORT:$DB_SID
```

Follow these steps if you are using a Windows operating system:

```bash
cd %RCU_SHIPHOMELOC%\bin
set DB_HOST=localhost
set DB_PORT=1521
set DB_SID=XE
set CONNECT_STRING=%DB_HOST%:%DB_PORT%:%DB_SID%
```

Example 2-3  Dropping the repository
Run this command if you are using a Linux operating system:

```bash
./rcu -silent -dropRepository -connectString $CONNECT_STRING -dbUser sys -dbRole sysdba -lockSchemas false -schemaPrefix SH -component SOAINFRA -component MDS - component ORASDPM -component BAM
```

Run this command if you are using a Windows operating system:

```bash
rcu.bat -silent -dropRepository -connectString %CONNECT_STRING% -dbUser sys -dbRole sysdba -lockSchemas false -schemaPrefix SH -component SOAINFRA -component MDS - component ORASDPM -component BAM
```

Example 2-4  Recreating the repository
Run this command if you are using a Linux operating system:

```bash
./rcu -silent -createRepository -connectString $CONNECT_STRING -dbUser sys -dbRole sysdba -lockSchemas false -schemaPrefix SH -component SOAINFRA -component MDS - component ORASDPM -component BAM
```

Run this command if you are using a Windows operating system:

```bash
rcu.bat -silent -createRepository -connectString %CONNECT_STRING% -dbUser sys - dbRole sysdba -lockSchemas false -schemaPrefix SH -component SOAINFRA -component MDS - component ORASDPM -component BAM
```

2.3.3 How to Set Up the Personal Environment for Standalone WebLogic Server
Following the steps in this section will give you a portable working environment without needing access to the full environment until you are ready to deploy and unit test.

While the JDeveloper-based environment with Integrated WebLogic Server is useful in creating and validating customizations to the Oracle ADF artifacts, it cannot be used to validate SOA customizations. Anything that relies upon SOA infrastructure for
development (such as BPM and ESS) will need the standalone environment in the following situations:

- Rapid testing of development before publishing to the full development environment.
- Remote access using either dial-up or VPN proves either to be too slow or unstable to connect to the full environment for successful deployment will benefit from the standalone environment installed locally on the developer's machine (64bit required).

To create this environment, you run Python scripts that are part of the JDeveloper-based environment and the repository of installer files that are used to create the shared environment. You first will have to create the JDeveloper-based environment, create or update the `fusion_apps_wls.properties` file, and then execute the scripts that are packaged with the `fa_dev_bundle.zip` extension bundle to create the standalone environment. One of the inputs to the Python script is the location of the repository that contains the installer files. You can use the Oracle Fusion Domain wizard from the JDeveloper-based environment to update the `fusion_apps_wls.properties` file such that it can be used in the standalone environment. So, the same properties file can be used to create both the JDeveloper-based environment and the standalone environment.

When the Python scripts are executed, they automatically do the following:

- Create a lightweight `MW_HOME` directory that is a subset of the Middleware home that is available in the shared environment.
- Create a Standalone WebLogic Server domain with an AdminServer and a ManagedServer.
- Use the `domainType` defined in the `fusion_apps_wls.properties` file to apply appropriate WebLogic Server templates and set up system properties for Oracle Fusion applications.
- Configure the domain with the data sources and the Identity Store that are available in the shared environment.
- Configure the domain to either point to the LDAP-based Policy Store in the shared environment or a local XML-based Policy Store.

Typically in this environment, you have to deploy the exploded EAR directory of the application from the Middleware home of the shared environment using the WebLogic Server Console. Consequently, the `adf-config.xml` descriptor contains an MDS metadata store that points to the database in the shared environment and the customizations are picked up from the MDS repository. If you have created customizations on the filesystem using the JDeveloper-based environment, you should import those customizations to the MDS schema in the database that is running as part of the shared environment to test and validate them. When you import the customizations to the repository and database in the shared environment, it will affect all the developers who are using the shared environment. You will be able to test and validate the customizations by exercising all the applications that have a touch-point with the customized application, to ensure that things outside the application are working as expected.

Because Standalone WebLogic Server for SOA points to a separate SOAINFRA MDS schema, the customizations need to be exported and imported into the shared environment when they are successfully tested by developers.
See "Managing the Metadata Repository" in the *Administering Oracle Fusion Middleware*.

Even though the Standalone WebLogic Server domain that is created as part of this environment is used to deploy the application from the same APPL_TOP and is configured to point to the same data sources, the Identity Store, and the Policy Store, as the domains that are part of the shared environment, you have the flexibility in setting up the domain that is part of the standalone environment in a way that it can work on a laptop or desktop without requiring excess resources. The domains that are created in the shared environment by the provisioning processes have one AdminServer and three ManagedServers. But, the domain in the standalone environment has just one AdminServer and one ManagedServer. You can decide whether to target SOA or ESS or various technologies at the ManagedServer based on the project.

**2.3.3.1 Create a Domain for Standalone WebLogic Server**

Installer files are used to create and run Standalone WebLogic Server domains. You may need to obtain the files from an administrator.

---

**Note:**

- The installer repository on Windows *must* be a local or a mapped drive.
- Windows and Linux operating systems *must* be 64-bit.
- You are using the 64-bit version of the JDK.

---

This install also allows Oracle SOA Suite developers to create their domains without extra installs or steps.

**2.3.3.1.1 How to Create a Special SOAINFRA Schema**

If you are creating a SOA customization, a special SOAINFRA schema that is in the database in the shared environment may need to be created so your work does not interfere with the normal database.

Because the existing composites reference Web Service Description Language (WSDL) and schemas in MDS, when new SOAINFRA and MDS_SOA schemas are created for the standalone environment, all the WSDLs and schemas needed by the composites to be customized need to be exported from the shared MDS_SOA and imported into the new standalone MDS_SOA schema.

See "Managing the Metadata Repository" in the *Administering Oracle Fusion Middleware*.

**2.3.3.1.2 How to Set Up the Environment for Standalone WebLogic Server**

Follow these steps to create a Standalone WebLogic Server environment. These steps assume that you already have downloaded and installed JDeveloper and the Fusion Apps Development Environment extension bundle.

1. If you have not already done so, download and install the 64-bit JDK, preferably not to a directory location containing spaces, such as \Program Files.

   When installing JDeveloper, you will select the 64-bit JDK location you just installed.
If a 32-bit JDK already was installed and used, you must delete the old %MW_HOME% installation and the system directory in %JDEV_USER_HOME% before reinstalling JDeveloper and the Oracle Fusion Applications extensions updates.

If you are using Windows XP and encounter issues running Integrated Weblogic Server, you will need to change to the 64-bit version of Windows 7.

2. You will need installer files from the provisioned environment. The site administrator should make the installer files available. When the system administrator downloaded the Oracle Fusion Applications DVDs and unzipped them, the installer files were placed in the installers directory that now contains numerous sub-directories. A directory listing looks similar to Figure 2-42.

![Figure 2-42  Installer Directory of a Provisioned Environment](image)

If you are not able to access a provisioned environment on a server, you will need to copy these directories (defined in the FADevInstallMwHome.py script file) and their contents from the installation disk to the installerLocation described in Table 2-3. (If you do not have the disk, you will need to download the entire 60GB Oracle Fusion Applications package from eDelivery.)

- atgpf
- biappsshiphome
- bishiphome (if included on the release)
- odi
- soa
- wc
- weblogic
- jdk

This subset of the entire installation package is approximately 20GB in size, with biappsshiphome using approximately 10GB.
Caution:
The installers directory is platform-specific. So if your production environment is Linux and you want to install Standalone WebLogic Server on a Windows machine, you will need to get the subset of installers from a Windows Oracle Fusion Applications download.

3. Update the properties file that was created for Integrated WebLogic Server so that it can be used to create Standalone WebLogic Server:

- In the Integrated WebLogic Server environment you already created, start JDeveloper, select the Oracle Fusion Applications Developer role, and launch the Oracle Fusion Domain wizard, as described in How to Use the Oracle Fusion Domain Wizard.

- Select the Standalone Server option on the wizard’s Usage dialog. Selecting this option creates or updates the fusion_apps_wls.properties and cwallet.sso files.

- On the Domain dialog, set Installer Location to the directory that the administrator has provided. See Step 2.

  Set Domain Location appropriately, such as /path_to_domain/FAStandaloneDomain. Make sure that the directory name you enter does not exist.

4. Create the Standalone WebLogic Server environment:

- Set these environment variables:

  csh commands:

  ```
  setenv MW_HOME /path_to/FAStandalone_MW_HOME
  setenv JDEV_MW_HOME /path/to/JDeveloper/install_directory
  setenv ANT_HOME $JDEV_MW_HOME/jdeveloper/ant
  setenv JAVA_HOME /path/to/JDK/installation/directory
  ```

  bash commands:

  ```
  export MW_HOME=/path_to/FAStandalone_MW_HOME.
  export JDEV_MW_HOME=/path/to/JDeveloper/install_directory
  export ANT_HOME=$JDEV_MW_HOME/jdeveloper/ant
  export JAVA_HOME=/path/to/installed/JDK
  ```

  Windows command prompt commands:

  ```
  set MW_HOME=%path_to\FAStandalone_MW_HOME
  set JDEV_MW_HOME=%path\to\JDeveloper\install_directory
  set ANT_HOME=%JDEV_MW_HOME%\jdeveloper\ant
  set JAVA_HOME=C:\JDK\install\directory
  ```

- `mkdir /path_to/FAStandaloneWork`

- `[for *nix only] chmod +x $JDEV_MW_HOME/jdeveloper/fadev/bin/* .py`

- `[Windows]`

  ```
  %JDEV_MW_HOME%\jdeveloper\fadev\bin\FADevInstallMwHome.py
  -m %MW_HOME%
  -i \path\to\installer_files
  ```
• Create a lightweight \texttt{MW\textunderscore HOME} directory by running the \texttt{FADevInstallMwHome.py} script. This script is installed when you install the Fusion Apps Development Environment extension bundle, described in Step 6 of \textit{Setting Up the JDeveloper-based Development Environment}. Note that the options have been placed on separate lines for clarity. When you run the script, all must be on the same line.

Run this command if you are using a Linux operating system:

```bash
$JDEV\_MW\_HOME/jdeveloper/fadev/bin/FADevInstallMwHome.py
-m $MW\_HOME
-i /path/to/installer\_files
-w /path/to/stagingdirectory
-v
```

Run this command if you are using a Windows operating system:

```bash
%JDEV\_MW\_HOME%\jdeveloper\fadev\bin\FADevInstallMwHome.py
-m %MW\_HOME%
-i C:\path\to\installer\_files
-w C:\path\to\stagingdirectory
-v
```

• \texttt{-m <mw\_home>}: Use an \texttt{MW\_HOME} directory other than the default, for an initial install or after it has already been created.

• \texttt{-i}: The location of the installer files. This setting is required only for a Standalone WebLogic Server domain and overrides the installerLocation setting in the properties file.

• \texttt{-w}: Set the working/staging directory for log and other temporary files.

• \texttt{-r}: Reinstall the \texttt{MW\_HOME} directory. Note that this will remove all items in the existing \texttt{MW\_HOME} directory.

• \texttt{-v}: Turn on verbose mode.

5. Patch the \texttt{MW\_HOME} directory.

For the standalone domain creation to succeed, you must patch these directories in the standalone \texttt{MW\_HOME} directory using the patches from the installer repository before executing the script to create the standalone domain. Otherwise, the standalone domain creation will not be complete and trying to deploy Oracle Fusion applications to the standalone domain will result in issues.

• atgpf

• biappsshiphome

• bishiphome (if included on the release)

• odi

• soa

• webcenter

• oracle_common
• Smart Update for Oracle Weblogic Server

Follow these steps to patch the directories in the standalone MW_HOME directory using all the opatches that are in the repository. The example uses the atgpf directory. Except for the Smart Update for Oracle Weblogic Server, replace atgpf in the opatch command with the names of the other directories, such as odi and soa.

Linux system

% setenv ORACLE_HOME $MW_HOME/atgpf
% setenv ATGPF_ORACLE_HOME $ORACLE_HOME
% setenv JHOME $MW_HOME/jdk6 [Note: Must be a 64-bit JDK.]
% setenv INV_LOC $ORACLE_HOME/oraInst.loc
% setenv PATH $ORACLE_HOME/OPatch:$PATH

% opatch napply <installer_repository_location>/installers/atgpf/patch -jdk $JHOME -invPtrLoc $INV_LOC

Repeat, substituting the names of the other directories in place of atgpf.

For the Smart Update, first change to the bsu directory:

% cd installation_directory/utils/bsu

Then use a command similar to the following, which installs the patch IRZ2 from the patch download directory that has been established for the current system:

% bsu -prod_dir=Oracle/Middleware/wlserver_10.3 -patchlist=IRZ2 -verbose -install

When executed, the preceding command displays the following output showing that the patch was successfully applied:

Checking for conflicts.
No conflict(s) detected

Starting installation of Patch ID: IRZ2
Installing /Oracle/Middleware/download-dir/IRZ2.jar
Result: Success

Windows system

c:\> set ORACLE_HOME=%MW_HOME%\atgpf
c:\> set ATGPF_ORACLE_HOME=%ORACLE_HOME%
c:\> set JHOME=%MW_HOME%\jdk6 [Note: Must be a 64-bit JDK.]
c:\> set PATH=%ORACLE_HOME%\OPatch;%PATH%

opatch napply <installer_repository_location>\installers\atgpf\patch -jdk %JHOME%

where -jdk is the location of the JDK under Oracle Fusion Middleware home.

[Note: For Windows, do not specify the invPtrLoc command-line argument.]

Repeat, substituting the names of the other directories in place of atgpf.

For the Smart Update, first change to the bsu directory:

cd c:\installation_directory\utils\bsu

Then use a command similar to the following, which installs the patch IRZ2 from the patch download directory that has been established for the current system:

bsu -prod_dir=c:\Oracle\Middleware\wlserver_10.3 -patchlist=IRZ2 -verbose -install
6. Create, extend and configure the Standalone WebLogic Server domain by running the FADevCreateDomain.py script. Note that the options have been placed on separate lines for clarity. When you run the script, all must be on the same line.

Run this command if you are using a Linux operating system:

```
$JDEV_MW_HOME/jdeveloper/fadev/bin/FADevCreateDomain.py
-r $MW_HOME
-p $JDEV_MW_HOME/jdeveloper/system11.1.1.7.xx.yy.zz/o.jdevimpl.rescat2/
fusion_apps_wls.properties
-i /path/to/installer_files
-w /path_to/FAStandaloneWork -v
```

Run this command if you are using a Windows operating system:

```
%JDEV_MW_HOME%\jdeveloper\fadev\bin\FADevCreateDomain.py
-r %MW_HOME%
-p %JDEV_MW_HOME%\jdeveloper\system11.1.1.7.xx.yy.zz\o.jdevimpl.rescat2\fusion_apps_wls.properties
-i \path\to\installer_files
-w \path_to\FAStandaloneWork
-v
```

**Example 2-5  FADevInstallMwHome.py Options**

Valid options are:
- **-m** : MW_HOME (standalone - will be created if it does not exist)
- **-i** : Installer location (the provisioning repository)
- **-w** : Working/staging directory. Used for response files, unzip installers
  Defaults to the current directory
- **-r** : reinstall MW_HOME
- **-v** : verbose

**Example 2-6  FADevCreateDomain.py Options**

- **-p** : property file
- **-m** : MW_HOME (standalone - will be created if does not exist)
- **-i** : installer location (Provisioning repository)
  overrides installerLocation in the properties file
- **-w** : working/staging directory for log and other temp files
- **-v** : verbose

**Example 2-7  FADevCreateDomain.py options**

If you execute FADevCreateDomain.py -help, the help shown in Example 2-6 will be displayed.

- **-p** <properties file>: Use a different fusion_apps_wls.properties file to configure the domain.
- **-m** <mw_home>: Use an MW_HOME directory other than the default, for an initial install or after it has already been created.
- **-i**: The location of the installer files. This setting is required only for a Standalone WebLogic Server domain and overrides the installerLocation setting in the properties file.
- **-w**: Set the working/staging directory for log and other temporary files.
- **-v**: Turn on verbose mode.
2.3.3.1.3 How to Fix Domain Creation Errors

These three causes of a domain creation error are easily corrected:

- The domain creation may fail in FADevCreateDomain.py related to the properties file such as:

  Creating domain for Fusion Applications Development ...
  ... checkWorkingDir
  ... checkPropFile
  Traceback (most recent call last):
  File "C:\Oracle\Middleware\jdeveloper\fadev\bin\FADevCreateDomain.py", line 546, in <module>
  checkPropFile()
  File "C:\Oracle\Middleware\jdeveloper\fadev\bin\FADevCreateDomain.py", line 146, in checkPropFile
  defaultPropFile= os.path.join(os.getenv('HOME'),'fusion_apps_wls.properties')
  File "C:\Python27\Lib\ntpath.py", line 96, in join
  assert len(path) > 0
  TypeError: object of type 'NoneType' has no len()

  For an Integrated WebLogic Server domain, this is caused by a Python installation issue. The suggested fix is to uninstall and re-install Python.
  For a Standalone WebLogic Server domain, the fix is to set the HOME environment variable. The recommended setting is to the user's home directory.

- The domain creation fails with:

  File "C:\Oracle\Middleware\jdeveloper\fadev\bin\FADevFusionAppsDomain.py", line 1754, in targetSharedLibraries
  KeyError: oracle.appstrace.model

  There is a synchronization issue between the domain creation scripts and templates used. This can be fixed by adding the entries tagged with "WORKAROUND" in the FADevFusionAppsDomain.py:

  'oracle.appltest.diagfwk.executor':14,   # ess-soa-adf
  'oracle.appstrace.model':2,              # WORKAROUND
  'oracle.appstrace.webapp':2,             # WORKAROUND
  'oracle.bi.adf.model.slib':15,           # ess-soa-adf-admin

- The domain creation can fail with:

  com.oracle.cie.domain.script.jython.WLSTException:
  com.oracle.cie.domain.script.jython.WLSTException:
  Error: unable to parse "template-info.xml" from template jar
  "c:\standalone\oracle_common\common\templates\applications\oracle.clickhistory_template_11.1.1.jar"

  Check whether the oracle.clickhistory_template_11.1.1.jar file exists. If it does not exist, copy it from JDEV_HOME to the standalone home:

  C:\Oracle>copy \oracle\Middleware\oracle_common\common\templates\applications\oracle.clickhistory_template_11.1.1.jar \standalone\oracle_common\common\templates\applications

2.3.3.1.4 How to Test the Server

To test the server:
1. Start the server using a command line command similar to:

[Windows]
C:\standalone\fusion_domain\bin\startWebLogic.cmd

[*nix]
/standalone/fusion_domain/bin/startWebLogic.sh

2. Once started, it can be accessed from a web browser, using a URL similar to:

http://1.23.45.678:7011/console

2.3.3.1.5 How to Manage the Standalone WebLogic Server Lifecycle

There will be times when you want to change the properties in fusion_apps_wls.properties, or point to a different Identity Store, or you may want to delete the domain and start from scratch. To do these, you will have to stop the running server, remove the domain directory, edit the properties file using the wizard, and recreate the domain. Follow these steps to accomplish the tasks. Remember to change any example directory names to the names you have used.

1. Stop the server

When you stop the server, use the same shell that was used to create the domain and execute these commands:

From Linux

<path_to_stopManagedWeblogic.sh>/stopManagedWeblogic.sh

If the script does not succeed, you will need to kill the process:

ps
kill -9 <pid_of_startWebLogic.sh> <pid_of_java>

From Windows

<path_to_stopManagedWeblogic.cmd>\stopManagedWeblogic.cmd

If you had started the ManagedServer, you should kill it, too.

2. Remove the domain

You may want to start over by removing the domain. Execute the following commands in a shell window:

rm -rf /path/to/FAStandaloneDomain
rm -rf /path/to/FAStandaloneWork/*

Execute these commands from a command prompt if you are using a Windows operating system:

rmdir /S /Q \path\to\FAStandaloneDomain
rmdir /S /Q \path\to\FAStandaloneWork/*

3. Edit the fusion_apps_wls.properties file

There may be times when you have to use a different Identity Store or modify some properties. In such an event, restart JDeveloper and follow these steps:
• Manually launch the Oracle Fusion Domain wizard. See How to Use the Oracle Fusion Domain Wizard.

• Right-click the Integrated Servers node and select the Configure Fusion Domain... option.

• Select the Standalone Server option from the first wizard dialog.

• Continue through the wizard, changing the property values as necessary.

4. Recreate the domain

To do so, execute these commands. Note that the options have been placed on separate lines for clarity. When you run the FADevCreateDomain.py script, all must be on the same line.

Run these commands if you are using a Linux operating system:

```bash
rm -rf /path/to/FAStandaloneDomain
rm -rf /path/to/FAStandaloneWork/*
$JDEV_MW_HOME/jdeveloper/fadev/bin/FADevCreateDomain.py
    -m $MW_HOME
    -p $JDEV_MW_HOME/jdeveloper/system11.1.1.7.xx.xx.xx/o.jdevimpl.rescat2/fusion_apps_wls.properties
    -i /path/to/Repository/installers
    -w /path/to/FAStandaloneWork -v
```

Run these commands if you are using a Windows operating system:

```bash
rmdir /S /Q \path\to\FAStandaloneDomain
rmdir /S /Q \path\to\FAStandaloneWork/*
%JDEV_MW_HOME%\jdeveloper\fadev\bin\FADevCreateDomain.py
    -m %MW_HOME%
    -p %JDEV_MW_HOME%\jdeveloper\system11.1.1.7.xx.xx.xx\o.jdevimpl.rescat2\fusion_apps_wls.properties
    -i \path\to\Repository\installers
    -w \path\to\FAStandaloneWork
    -v
```

2.4 Use Deployment Profiles Settings

When creating an Oracle ADF library deployment profile, you can include connection information. When a project attaches that Oracle ADF library, the connection information is merged with its own connection information. This provides runtime consistency. The Oracle ADF library, by including the connection information, can ensure that all of the resources that it needs (the connections) are properly propagated to the consumers.

When creating an Oracle ADF library deployment profile, the default is to include all connection details for every connection in the connections.xml, which is a workspace level file. Subsequently, when the Oracle ADF library is attached to a project, all of the connections are merged with the connections.xml for that project's workspace. This causes a proliferation of the connections across Oracle Fusion Applications. While the propagation of the connections is desirable, it is propagating much more than is really needed.

**Example of Connections Propagation**

A Financials project creates an Oracle ADF library with the defaults. All of the connection information for that Financials workspace is included in the Oracle ADF
library. HCM picks up that Oracle ADF library. HCM’s workspace now contains all connections that HCM needs, and all of the connections from the Financials workspace. If the defaults are retained, all of HCM’s projects contain connection information from Financials plus HCM. If CRM picks up any of those HCM Oracle ADF libraries, it merges the connection information into the CRM workspace; which now contains all of Financials plus HCM plus CRM.

2.4.1 How to Use Service Deployments

A project that contains an Oracle ADF Business Components-based service can have two purposes. The Oracle ADF Business Components code can be invoked as a service or it can be used as a regular Oracle ADF Business Components object. Oracle ADF provides two different deployment profiles to handle each of these cases.

For the service scenario, the BC Service Profile creates two JAR files. One is the **Common** one that contains information that is needed by the service invoker (Web Service Description Language (WSDL), XML schema definition (XSD), Service Interface). The **MiddleTier** one is an Enterprise JavaBeans (EJB) JAR file that contains the actual implementation.

For use as an Oracle ADF Business Components object, consumers must get an Oracle ADF library. That is the only way the Oracle ADF Business Components objects are exposed to consumers in the Oracle ADF Business Components design time wizards. Oracle ADF library also has no option for filtering, so it includes all the artifacts from the project including the WSDL, XSD, and Service Interface. Additionally, the Oracle ADF library includes the connection information for invoking the service. Because of this, developers inherit extra connection information if they want to use a service-enabled application module, not as a service, but as an application module.

**Common** must be an Oracle ADF library because consumers of this need a connection entry to be injected into the consumers’ `connections.xml` file. This does not happen with ordinary JAR files.

2.4.2 How to Update the Standard

All Oracle ADF library deployment profiles should be updated to selectively include connections that are important to that one project. Common scenarios include:

- **Data Model Project**
  - ApplicationDB database connection
  - ApplicationsRepository, if you use Attachments
  - Service connections for any ServiceFactory invocations
  - Essbase
- **User interface project**
  - Portlet producers
  - Web Service Data Control connections

In the **Edit ADF Library JAR Deployment Profile Properties** dialog, choose to include **Connection Name Only**, as shown in **Figure 2-43**
2.5 Configure the Oracle Enterprise Scheduler (ESS)

If you have elected to use Standalone WebLogic Server for Oracle Fusion Applications ESS development, you will need to perform the tasks in this section.

For information about using the Oracle Enterprise Scheduler, see the Developing Applications for Oracle Enterprise Scheduler.

**Note:**

The ESS and Fusion schema *must* be located in the same database and *must* be linked to each other.

Provisioning should set up all the required synonyms and grants between the schema. The FUSION schema has views, such as ESS_REQUEST_HISTORY, that are linked to the ESS schema. To validate that some linking is in place, check to see if the view works. (Although this is not a complete check of all links, it will fail if no linking has been established.)

2.5.1 How to Provision the Runtime Environment

*How to Set Up the Personal Environment for Standalone WebLogic Server* shows how to configure the fusion_apps_wls.properties file and run the FADevConfigDomain.py script to stage and deploy the necessary infrastructure. For Oracle Enterprise Scheduler, you must ensure that you have configured the `domainType` property in the `fusion_apps_wls.properties` file to `standalone`, `adminall` or `adminessadf`. Example 2-8 shows correctly configured Oracle Enterprise Scheduler database and schema information, and ESS-related settings.

After provisioning the runtime, you should create the supporting database schema before starting the managed servers which is covered in *How to Create Supporting Database Schema.*
Example 2-8  Sample Showing Correctly-Configured Oracle ESS Database and Schema Information, and ESS-related Settings

[domain]
domainType=adminessadf
domainName=fusion_domain
listenPort=7011
soaPort=7012
wlName=weblogic
wlPassword=password
...
[wlsconfig]
fusionDbHost=host.example.com
fusionDbPort=1522
fusionDbSid=fusiondbsid
...
# leave oraessDbHost, oraessDbPort, oraessDbSid blank if using fusion database
oraessDbHost=
oraessDbPort=
oraessDbSid=
oraessDbUser=oraess_dbuser
oraessDbPassword=password
#
essMdsDbHost=
essMdsDbPort=
essMdsDbSid=
essMdsDbUser=fusion_mds_ess
essMdsDbPassword=password

2.5.2 How to Create Supporting Database Schema

There are two approaches to creating the database schema: using the Oracle Fusion Applications Repository Creation Utility or creating the schema using SQL scripts. The latter allows greater flexibility in the naming of the schema and user, but requires use of SQL*Plus.

For pre-requisite steps and configuration of the Oracle Enterprise Scheduler schema using the Oracle Fusion Applications Repository Creation Utility, see How to Use Alternate Database Schemas. Example 2-9 shows how to configure the Oracle Enterprise Scheduler schema using the Oracle Fusion Applications Repository Creation Utility schema.

Alternatively, creating the schema by running scripts in SQL*Plus can be performed as shown in Example 2-10

---

**Note:**

You should determine the appropriate TEMP tablespace by reviewing the entries in dba_tablespaces before attempting to run these scripts.

---

Example 2-9  Configuring Oracle Enterprise Scheduler Schema Using the Oracle Fusion Applications Repository Creation Utility Schema

Run these commands if you are using a Linux operating system:

```bash
cd $RCU_SHIPHOMELOC/bin
DB_HOST=localhost
DB_PORT=1521
DB_SID=XE
```
CONNECT_STRING=$DB_HOST:$DB_PORT:$DB_SID

./rcu -silent -createRepository -connectString $CONNECT_STRING -dbUser sys -dbRole sysdba -lockSchemas false -schemaPrefix D8B2 -component ESS

Run these commands if you are using a Windows operating system:

cd %RCU_SHIPHOMELOC%\bin
set DB_HOST=localhost
set DB_PORT=1521
set DB_SID=XE
set CONNECT_STRING=%DB_HOST%:%DB_PORT%:%DB_SID%

rcu.bat -silent -createRepository -connectString %CONNECT_STRING% -dbUser sys -dbRole sysdba -lockSchemas false -schemaPrefix D8B2 -component ESS

Example 2-10 Using SQL*Plus Scripts to Create Schema

cd $MW_HOME/rcu/rcu/integration/ess/sql
sqlplus sys/manager as sysdba;
@createuser_ess_oracle.sql oraess_d8b2 oraess_d8b2 SYSTEM TEMP;
connect oraess_d8b2/oraess_d8b2
@createschema_ess_oracle.sql  oraess_d8b2

2.5.2.1 Post-Installation Checks

Perform these steps to make sure the ESS installation was successful.

2.5.2.1.1 Verifying the Temp Directory Location and Write Permissions

The ESSAPP (also known as the ESS Base application) is the deployed infrastructure that supports the deployment of the product team Oracle Enterprise Scheduler applications, known as hosted applications. By default, this application writes all request log and output to a directory path known as the \userFileDir\, which is configured in the \ess.xml\ file.

The ESS application defaults to file persistence mode and writes all the request log and output to a directory path known as the RequestFileDirectory, which is configured in the ESSAPP connections.xml file. By default, the temp directory will point to /tmp/ess/requestFileDirectory. Ensure that the directory exists and, if not, create it as the user who will start the ESS managed server.

2.5.2.1.2 Verifying ESS Artifacts Deployment Targets

Make sure ESS datasources and shared libraries are targeted to clusters and managed servers. Stop and Start WebLogic Servers as needed.

2.5.2.1.3 Checking ESS Health

Run ESS Health checks by accessing these links:

- Checking health of an ESS Node: http://<hostName>:<port>/EssHealthCheck/checkHealth.jsp
- Checking health of an ESS Cluster: http://<hostName>:<port>/EssHealthCheck/diagnoseHealth.jsp
2.6 Test Your Installation

To test your JDeveloper and Oracle ADF installation, perform the following steps to create both a data model project and a user interface project, create an ApplicationDB database connection, and create and run a simple page.

1. In JDeveloper, select the Application Navigator menu, then select New Application to open the Create Application wizard. See Figure 2-44.

**Figure 2-44  Naming Your Application**

![Create Fusion Web Application (ADF) - Step 1 of 5](image)

**Tip:**
The name of the wizard changes according to the application template that is selected.

2. Complete the following:
   - **Application Name:** Enter `Setuptest`
   - **Application Package Prefix:** Enter `oracle.apps.test`
   - **Application Template:** Choose `Fusion Web Application (ADF)`

3. Click Next to access the Name Your ADF-Model Project dialog. See Figure 2-45.

**Note:**
The system automatically will create data model and user interface projects for you. The default names for these projects that JDeveloper provides are `Model` and `ViewController`.

You can enter a new name for your data model project or you can keep the default name `Model`. 
Figure 2-45  Naming Your Oracle ADF Model Project

The Project Technologies are automatically selected based on the application template that was chosen. You can select additional technologies if required.

4. Click **Next** to access the Configure Java Settings for the ADF-Model dialog. This dialog displays the Java settings for your data model project. See Figure 2-46.

Figure 2-46  Configuring Java Settings for the ADF Model
5. Click Next to access the Name Your ADF ViewController Project dialog.

You can enter a new name for your user interface project or you can keep the default name ViewController. See Figure 2-47.

**Figure 2-47 Naming Your Oracle ADF User Interface Project**

![Create Fusion Web Application (ADF) - Step 4 of 5](Image)

**Note:**

The Project Technologies are automatically selected based on the application template that was chosen. You can select additional technologies if required.

6. Click Next to access the Configure Java Settings for the ViewController... dialog.

This dialog displays the Java settings for your user interface project. See Figure 2-48.
Click **Finish** to create your new application.

7. Add the **Applications Core, Applications Core (Attachments Model), Topology Manager, Functional Setup Model, BC4J Service Runtime**, and **Java EE 1.5** libraries to the data model project. See **Adding Necessary Libraries to Your Data Model Project**.

8. Add the **Applications Core (ViewController)** tag library to the user interface project. See **Adding the Applications Core Tag Library to Your User Interface Project**.

9. Add this option to the Model project's Run options:

   ```
   -Doracle.jdbc.createDescriptorUseCurrentSchemaForSchemaName=true
   ```

   **Note:** Setting this JVM parameter helps avoid an error caused by the username for the current schema being set to “fusion” instead of “fusion_runtime” Setting the username to “fusion” results in incorrect data type associations. This parameter sets the username back to “fusion_runtime.”

   To add the option:

   - Right-click the Model project and select **Project Properties**.
   - Select **Run/Debug/Profile**.
   - In the Run Configurations list, select **Default** and click **Edit**.
   - Select **Launch Settings**.
     As shown in **Figure 2-49**, enter the option string in the Java Options field.
• Click OK.
• Click OK.

10. The ApplicationDB database connection is created automatically when the Applications Core library is attached to the model project. However, you will need to adjust the values that are defined in the connection to reflect the database you want to use. See Creating or Updating a Database Connection. For this example, enter these connection details:

   **Connection Type:** Choose Oracle (JDBC)

   **Username / Password:** Enter the username and password for your team’s Fusion_Runtime database schema.

   **Deploy Password:** Select this option.

   **Host Name:** Enter the host name, such as my.host.com

   **JDBC Port:** Enter the port number for your database.

   **SID:** Enter the database name, such as mydb.

11. Choose Application Navigator > Model. Right-click and choose New from the menu to open the New Gallery.

12. Choose the Business Tier > ADF Business Components category. Select the Business Components from Tables item to launch the Create Business Components from Tables wizard. See Figure 2-50.
13. Complete the following to create your entity object:

a. **Filter Types**: Select only *Tables* to narrow your search for schema objects.

b. **Filter Name**: Enter a filter, such as `%EMP%` to narrow your search to tables.

c. **Query**: Click this button to perform your search.

---

**Note:**
The results of your search displays in the *Available* column.

---

d. Choose the required object, such as *FND_DEMO_EMP* and click > to shuttle it over to the *Selected* column.

e. Click Next to go to the next step in the wizard.

14. Choose the required entity object located in the *Available* column and click > to shuttle it over to the *Selected* column. See Figure 2-51.
15. Click Finish to create your updatable view object and to close the Business Objects wizard.

16. Ensure that your application module configuration is using JDBC data source.
   
   This is required for your application module to run on the WebLogic Server.

   To update your application module configuration:
   
   **a.** Go to the Application Navigator and select your application module. Right-click and select Configurations from the menu.
   
   **b.** Choose the configuration <AM Name>Local, then choose Edit.
   
   **c.** Change the Connection Type to JDBC DataSource and Datasource Name to java:comp/env/jdbc/ApplicationDBDS. Click OK.

17. Validate your model with the Business Component Tester to make sure that the ApplicationDBDS data source has been configured for the Integrated WebLogic Server environment.

   In the Navigator tree, right-click the application module and select Run, as shown in Figure 2-52.

**Figure 2-51  Create Business Components from Tables — Updatable View Objects**

![Create Business Components from Tables - Step 2 of 6](image)

**Figure 2-52  Running the Application Module**

![Running the Application Module](image)
If your installation is set up correctly, a dialog similar to that shown in Figure 2-53 displays. If an error message displays, you will need to re-check that the previous steps have been performed correctly.

**Figure 2-53  Application Module in Business Component Browser**

In this example, right-click FndDemoEmpView1 and select *Show* to display data, as shown in Figure 2-54.

**Figure 2-54  Showing Data in the Business Component Browser**

18. Choose *Application Navigator > ViewController*. Right-click and select *New* from the menu to open the New Gallery.

19. Choose the *Web Tier > JSF* category. Select the *JSF Page* item and click *OK* to open the Create JSF Page dialog.

20. Complete the following:
   - **File Name**: Enter *Setup.jsp*
   - Select to create jspx file.
   - Click *OK*.

21. Go to the Data Controls panel and drag the collection onto the open *Setup.jspx* window. See Figure 2-55.
22. Select to create Forms > ADF read-only from the context menu that displays. See Figure 2-56.

23. Remove some of the rows that are displayed in the opened dialog so that your page only lists a few fields. Select the Include Navigation Controls checkbox and click OK. See Figure 2-57.
24. Click the Run button located on the toolbar to run your page.

**Note:**

Make sure the URL uses the full host name. For instance, if the displayed URL is `http://127.0.0.1:7101/AppICoreCRMDemo/faces/Region6UIShellPage`, you should edit it manually so it appears similar to `http://myhost.name.com:7101/AppICoreCRMDemo/faces/Region6UIShellPage`.

When your page is displayed, you can use the buttons that appear at the bottom of your page to view next and previous employees.

### 2.7 Use Best Practices for Setting Up the Development Environment

Implementing these best practices when using JDeveloper will significantly reduce problems.

#### 2.7.1 How to Implement Best Practices for JDeveloper

These recommendations are specific to improving the performance of JDeveloper.

**Increase the Number of Lines in the Log Message Window**

The default of 3000 lines generally is insufficient for Oracle Fusion applications, and important errors and exceptions may be removed too quickly. The solution is to increase the number of lines, such as to 30000. Whenever you create a new view and run JDeveloper for the first time, increase the limit. Open **Tools > Preferences > Environment > Log** and edit the **Maximum Log Lines** setting.
Running JDeveloper in Verbose Mode

You can run JDeveloper in its default non-verbose mode, or in its verbose mode.

- **jdev**: The default non-verbose mode limits the amount of information displayed to the console. This helps you focus on the important information being displayed.

- **jdev -v**: The verbose mode displays all the information to your console. Although this information is not useful for everyday workflow, when something goes wrong, more information can help you debug your problem.

Increase the minimum / maximum heap size for JDeveloper (and other Java parameters)

This is specifically about increasing the heap size for JDeveloper, since JDeveloper itself is a Java executable and runs in its own Java Virtual Machine (JVM). This will not affect Integrated WebLogic Server; for that you set USER_MEM_ARGS, since it's a separate process and therefore a separate JVM.

To change the values for minimum and maximum Java heap, modify the corresponding parameters in `$jdev_install/ide/bin/ide.conf`.

Other parameters can be set in `$jdev_install/jdev/bin/jdev.conf`.

Do not set Xms or Xmx in the `jdev.conf` file because it will just result in duplicating the parameter on the command line because it already is set in the `ide.conf` file. You can add any other parameter than is not already passed on the command line in this file, using the same format as the existing parameters.

Enable the JDeveloper Java heap meter

You can enable the JDeveloper heap monitor (that is, the heap, permgen, and dustbin icon that forces garbage collection on the status bar of the main jdev window). Add this line to the `$jdev_install/jdev/bin/jdev.conf` file.

```
AddVMOption -DMainWindow.MemoryMonitorOn=true
```

The heap monitor shows the current size of the heap; not necessarily the maximum size. The heap is created at the specified minimum size. When additional space is required, and if garbage collection cannot free up enough space, the heap size is increased. If the heap reaches its maximum and there still is not enough space after garbage collection, an OutOfMemoryException is thrown.

2.7.2 How to Refresh the Oracle ADF Library Dependencies Library

The Oracle ADF Library Dependencies library is refreshed by doing the Refresh ADF Library Dependencies.

- There is a new library file per project in the project directory. This file will only exist if the project has unresolved deployment dependencies required by the directly-imported Oracle ADF JAR files in the project.

- The file should be added to the project source-controlled file set.

- The file should be included in all transactions where it was updated during the design time.

- If a runtime exception, such as No Def Found or No Class Def Found occurs, the project menu command to refresh the Oracle ADF Library Dependencies is
necessary to update the file. This could happen because of updated lower-level dependency changes outside of the design time session.

2.7.3 How to Manage OutOfMemory Exceptions (PermGen)

When you use Integrated WebLogic Server, make sure the USER_MEM_ARGS environment variable is set before starting JDeveloper.

USER_MEM_ARGS=-Xms256m -Xmx1024m -XX:MaxPermSize=512m -XX:CompileThreshold=8000

- The csh command is:
  `setenv USER_MEM_ARGS "-Xms256m -Xmx1024m -XX:MaxPermSize=512m -XX:CompileThreshold=8000"`

- The bash command is:
  `export USER_MEM_ARGS="-Xms256m -Xmx1024m -XX:MaxPermSize=512m -XX:CompileThreshold=8000"

Verify that it is set correctly.

```
$ env | grep USER_MEM_ARGS
```

$USER_MEM_ARGS is read by the WebLogic Server startup scripts, and is used to override the default JVM memory settings. If using the default MaxPermSize=256M, you will regularly get OutOfMemoryExceptions due to exhausted permGen. Setting permGen higher doesn’t completely fix the problem, but it does mean you can work longer before deployment fails with a permGen-related OutOfMemoryException.

In the JDeveloper message log window, you will see this line when Integrated WebLogic Server is started. Make sure it reflects the overridden values defined in $USER_MEM_ARGS.

`JAVA Memory arguments: -Xms256M -Xmx1024M -XX:CompileThreshold=8000 -XX:PermSize=64M -XX:MaxPermSize=512M`

Remember that overriding the Java memory arguments is a balancing act, and if you set them too high for your machine resources, either JDeveloper or WebLogic Server may fail to start, may hang, or may fail with a resource-related exception. For example, setting XX:MaxPermSize=1024m may be too high. If you experience problems after increasing the permanent generation size, try unsetting $USER_MEM_ARGS to see if it could be the cause.

Example exceptions

```
java.lang.OutOfMemoryError: PermGen space
at java.lang.ClassLoader.defineClass1(Native Method)
at java.lang.ClassLoader.defineClass(ClassLoader.java:621)
at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:124)
at weblogic.utils.classloaders.GenericClassLoader.defineClass(GenericClassLoader.java:344)
Truncated. see log file for complete stacktrace
```

Sometimes deployment just becomes very slow before it eventually fails.

When you hit OutOfMemoryExceptions, if you then try and close Integrated WebLogic Server, the first attempt may fail because it is in a bad state. If you try a second time, JDeveloper now does a `kill -9`, which should clear it. You should no longer need to kill WebLogic Server by manually issuing the kill command from another terminal session. However, if you ever do need to, try identifying the WebLogic Server process. This command assumes that you are using the default port 7101.
If you have another instance of WebLogic Server running, and the port is already in use, JDeveloper will use another port. Also, you may have changed the port in fusion_apps_wls.properties.

This command lists the Java processes with the full command line, which should help you to identify the WebLogic Server process.

```
$ ps -elf | grep java | grep <userid>
```

### 2.7.4 How to Work with Oracle ADF Libraries at Design Time

Every data model or user interface project should have an Oracle ADF library deployment profile. Service projects are the exception.

Oracle ADF libraries should be added to your project using the Resource Catalog by creating a File connection. From there, you can right-click any of the libraries and select **Add to Project**. Then all Oracle ADF libraries get managed under a Library called **ADF Libraries**. Mixing and matching different methods of adding Oracle ADF libraries can cause them to appear under different Libraries and sometimes under multiple libraries. That makes it hard to manage.

All references to components contained in Oracle ADF libraries are resolved when the workspace is loaded in JDeveloper. If a reference to a component or Java class in an Oracle ADF library cannot be resolved because, for instance, it does not exist or is incompatible with the existing reference, you probably will receive a compilation error.

Closing and restarting JDeveloper with a workspace open does not refresh the references to Oracle ADF libraries. Closing the workspace, and re-opening it does.

If you have a specific project selected in the JDeveloper navigator pane, select View > **Refresh ADF Library Dependencies** for *.jpr* to refresh the references to Oracle ADF libraries.

When you make any changes to the components in a project, where the components are being referenced as an Oracle ADF library by your user interface project, you need to redeploy the Oracle ADF library and refresh the Oracle ADF library dependencies for your user interface project. The same applies to one model project referencing from another model project.

If you are developing or debugging code in a data model project while running the referencing user interface project to test it, it may be easier to add the model project as a build output dependency, so you do not have to go through the cycle of redeploying the Oracle ADF library or refreshing Oracle ADF library references each time you make a change.
Set Up Your JDeveloper Application Workspace and Projects

This chapter describes how to set up your JDeveloper application workspace and projects, add libraries to projects, integrate Oracle Fusion Middleware extensions, create a database connection, implement Oracle Enterprise Crawl and Search (ECSF), and deploy Oracle SOA Suite.

Whenever you create new projects, you must first create an application using the Fusion Web Application (Oracle ADF) template. The system will then automatically create the data model and user interface projects for you. The default names that JDeveloper provides for these projects are Model and ViewController.

After your projects have been created, you must manually add the Applications Core library to the data model project and the Applications Core Tag library to the user interface project.

This chapter discusses:
- Using Technology Scopes
- Provisioning the Application Workspace
- Adding Necessary Libraries to Your Data Model Project
- Adding the Applications Core Tag Library to Your User Interface Project
- Integrating () Setup UIs
- Creating or Updating a Database Connection
- Adding the Search Navigation Tab to the Overview Editor for (ECSF)
- Overriding the Default Resource Bundle
- Implementing Oracle Enterprise Scheduler Service Workspace and Deployment
- Implementing Oracle Application Development Framework UI Workspace and Projects

3.1 Using Technology Scopes

Technology scopes are attributes on the project that can be used to identify the different technologies used for that particular project. These attributes are used only within JDeveloper to assist you as you work. With technology scopes, the choices presented to you in the New Gallery and in the menus and palettes are filtered so that you see only those items that are most relevant to you as you work. Technology scopes have no effect on the data in the project itself.

The JDeveloper online Help has more information.
3.2 Provisioning the Application Workspace

The application’s Enterprise Archive (EAR) will be available for developers to pick up when creating a custom application workspace. An EAR is a Java EE archive file that is used in deploying applications on a Java EE application server. Framework applications are deployed using both a generic EAR file, which contains the application and the respective runtime customization, and a targeted EAR file, which contains only the application for deployment to the application server. An administrator that provisions the environment will be responsible for providing developers with the following:

- EAR locations for the various applications.
- jazn-data for the various applications.
- LDAP/credential store that the developer can set up for authentication.

In addition, the Oracle Fusion Applications Customization Application Wizard will create a complete development environment for customizing existing Oracle Fusion applications. See the online Help in the wizard, and “Using Oracle JDeveloper for Customizations” in the Extensibility Guide for Developers.

3.3 Add Necessary Libraries to Your Data Model Project

Use these directions to add the Applications Core, Applications Core (Attachments Model), Topology Manager, Functional Setup Model, BC4J Service Runtime, Java EE 1.5, and Java EE 1.5 API libraries to the data model project. The default name, provided by JDeveloper, for this project is Model.

To add the necessary libraries to a data model project:

1. Choose Application Navigator > Model project. Right-click and choose Project Properties from the menu.

2. Choose the Libraries and Classpath category. Click Add Library to open the Add Library dialog.

3. Select the Applications Core, Applications Core (Attachments Model), Topology Manager, Functional Setup Model, BC4J Service Runtime, Java EE 1.5, and Java EE 1.5 API libraries from the list of available libraries. Click OK to save your selection and close the Add Library dialog.

The libraries are now displayed in the Classpath Entries region of the Libraries and Classpath dialog, as shown here.
4. Click OK to save your changes.

3.4 Add the Applications Core Tag Library to Your User Interface Project

Use these directions to add the Applications Core Tag Library to the user interface project. The default name provided by JDeveloper for this project is ViewController.

To add the Applications Core Tag library to the user interface project:

1. Choose Application Navigator > ViewController project. Right-click and choose Project Properties from the menu.

2. Choose the JSP Tag Libraries category. Go to the Distributed libraries folder and click Add to open the Add Library dialog.

3. Select the Applications Core (ViewController) 11.1.1.0.0 tag library from the list of available libraries. Click OK to save your selection and close the Add Library dialog.

The Applications Core (ViewController) 11.1.1.0.0 is now displayed under the Distributed libraries folder on the JSP Tag Libraries dialog, as shown in Figure 3-2.
4. Click **OK** to save your changes.

5. Choose **Application Navigator > ViewController** project. Right-click and choose **Project Properties** from the menu.

Choose the **Dependencies** category and select the **Model.jpr**.

---

**Note:**

Even if you are only using the user interface project you must still initialize the data model project as they are dependent on each other.

---

6. Click **OK** to save your changes and close the Project Properties dialog.

### 3.5 Integrating Oracle Fusion Middleware Extensions for Applications (Applications Core) Setup UIs

The most common use of Applications Core setup UIs is through Oracle Fusion Functional Setup Manager tasks that invoke the UIs running on the Applications Core Setup Java EE application. Applications Core setup UIs are part of the Applications Core (Setup UI) shared library, which is hosted centrally in the Applications Core Setup Java EE application. Consequently, developers typically will not need to include the shared library in their own Java EE applications.

#### 3.5.1 What You May Need to Know About Setup UIs in Oracle Fusion Functional Setup Manager

Every Oracle Fusion application registers ADF task flows with the Functional Setup Manager, which provides a single, unified user interface that allows implementers and administrators to configure all applications by creating set up data.
For example, a Human Resources application can register setup activities such as "Create Employees" and "Manage Employee Tree Structure." For more information, see the Oracle Fusion Functional Setup Manager User's Guide.

To make these task flows available to developers, implementers or administrators, a developer integrates the desired Applications Core setup UI task flows with Functional Setup Manager. For information about specific task flows, see:

- About Integrating Messages Task Flows into
- About Integrating SetID Task Flows into
- About Integrating Lookups Task Flows into
- About Integrating Document Sequence Task Flows into
- About Integrating Attachments Task Flows into
- Integrating Flexfield Task Flows into
- About Integrating Profiles Task Flows into

The most common use of setup UIs is through Oracle Fusion Functional Setup Manager tasks. This is true even for product-specific tasks that invoke the task flows with parameters that restrict the results to a single object or set of objects.

### 3.5.2 How to Integrate Setup UIs into Functional Setup Manager

To determine your requirements, familiarize yourself with these three scenarios and decide which one best fits your needs. The first two patterns are the typical use cases. The third is for approved exceptions only.

- **Scenario 1** is a generic setup task that invokes a setup task flow running in the Applications Core Setup Java EE application. For example, you want to give your product administrator roles access to the generic Manage Descriptive Flexfields setup task.

- **Scenario 2** is a product team-specific setup task that invokes a setup task flow running in the Applications Core Setup Java EE application and passes in product-specific parameters to restrict the objects to only those relevant to this specific task. For example, you want to give your product administrator roles access to a Manage GL Descriptive Flexfields setup task that launches the Manage Descriptive Flexfields setup UI for descriptive flexfields belonging to the GL module only.

- **Scenario 3** is a product team-specific setup task that invokes a setup task flow running in the product team's own Java EE application. This scenario is for approved exceptions only. For example, you plan to embed the setup UI within another UI in your own product team's Java EE application. For instance, the Manage Item Categories UI in Product Information Management (PIM) embeds the Manage Extensible Flexfields setup UI.

Follow the instructions in Table 3-1 that are relevant to your scenario to integrate Applications Core setup UIs into Functional Setup Manager.
Table 3-1  Instructions for Each Scenario

<table>
<thead>
<tr>
<th>Step</th>
<th>Scenario 1</th>
<th>Scenario 2</th>
<th>Scenario 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Follow Functional Setup Manager guidelines to create product-specific setup tasks in the Application Design Repository. Tailor the behavior of the setup UI by passing allowed values to the task flow parameters. Decide what Applications Core setup UI task flows that you want to incorporate and locate the chapter (see What You May Need to Know About Setup UIs in Oracle Fusion ) that describes each task flow and its parameter values.</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Product teams should set the value of the Enterprise Application field (in the Application Design Repository) to the appropriate Java EE application for any of their product-specific Functional Setup Manager tasks that use Applications Core setup task flows. Typically, this should be set to the Applications Core Setup Java EE application.</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Ensure product team roles inherit the appropriate Applications Core duty role. The duty roles support securing the setup tasks so only authorized users have access.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>If you intend to integrate a product-team specific setup UI and it will run in your product team's own Java EE application, your application will need to include the Applications Core shared library.</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>For any of the duty roles and their associated privileges that your application inherits, include permissions for those privileges in your application's jazn-data.xml file. Permissions make it possible to grant authorized users access to your setup tasks.</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

3.6 Creating or Updating a Database Connection

A connection to a valid database is necessary to run most, if not all, applications.

When you have added the libraries as described in Adding Necessary Libraries to Your Data Model Project and Adding the Applications Core Tag Library to Your User Interface Project to your project, the connections for the ApplicationDB, ApplicationAuditDB, and AppMasterDB databases will have been created. You just need to update them to have the connection information for your actual environment.

To create or update a database connection:

1. In JDeveloper, choose Application Resources > Connections. Right-click and choose New Connection > Database from the menu.

2. Add or update the following connection details for the ApplicationDB connection name as shown in Figure 3-3.
Figure 3-3 Create Database Connection Dialog

Connection Name: The value for the connection name must be ApplicationDB.

Connection Type: Choose Oracle (JDBC).

Username and Password: Enter the database username and password.

Deploy Password: Select this checkbox.

Host Name: This is the default host name if the database is on the same machine as JDeveloper. If the database is on another machine, type the name (or IP address) of the computer where the database is located.

JDBC Port: This is the default value for the port used to access the database. If you do not know this value, check with your database administrator.

SID: This is the default value for the SID that is used to connect to the database. If you do not know this value, check with your database administrator.

3. Click Test Connection. (Database listener Port). If the database is available and the connection details are correct, the message Success! is displayed. If not, review and correct the information that you entered.

4. Click OK. The connection now appears below the Application Resources Connections folder as shown in Figure 3-4.
3.7 Adding the Search Navigation Tab to the Overview Editor for Oracle Enterprise Crawl and Search Framework (ECSF)

ECSF provides developers a set of tools and a framework to quickly and efficiently integrate Oracle Secure Enterprise Search (SES) into enterprise applications to expose business objects for full text search.

Developers use ECSF to integrate search functionality in Oracle Fusion applications by defining searchable objects and searchable attributes. Defining searchable objects and searchable attributes enables the corresponding view objects and view object attributes for search, and creates the necessary metadata for ECSF. However, before you can define searchable objects and searchable attributes, you must add the Search navigation tab to the overview editor in JDeveloper.

For more information about defining searchable objects, see Creating Searchable Objects.

3.7.1 How to Add the Search Navigation Tab to the Overview Editor

To add the Search navigation tab to the overview editor in JDeveloper, download the JDeveloper extension for ECSF.

To download the JDeveloper extension for ECSF:

1. Launch JDeveloper.

Note:

If you have trouble initializing Java Virtual Machine (JVM), launch JDeveloper by entering jdeveloper.exe -J-Xmx125m at a command prompt.

2. In JDeveloper, choose Check for Updates from the Help menu.
3. In the Check for Updates dialog, click Next.
4. In the Source tab, select the Search Update Centers radio button, then select the Internal Automatic Updates checkbox.
5. Click Next, then select the ECSF Design Time Extension checkbox.
6. Click Next, then click Finish.
7. When prompted to restart JDeveloper, click Yes.
3.7.2 What Happens When You Add the Search Navigation Tab to the Overview Editor

When the Search navigation tab is added, the oracle.ecsf.dt.jar file appears in the oracle_home/jdev/extensions directory and the following files appear in the oracle_home/ecsf/lib directory:

- ecsfSchema.sql
- ecsfSysView.sql
- search_admin_wsclient.jar
- ecsf.jar
- ecsf-dt_bundle.zip
- search_client.jar
- ecsfSeedData.sql

The Search navigation tab appears in the overview editor of JDeveloper, as shown in Figure 29-3.

Use the Search navigation tab to configure the search-related properties.

For more information, see Creating Searchable Objects.

3.8 Overriding the Default Resource Bundle

In an Oracle Fusion application, strings are not hard-coded; they are placed in resource bundles. An application can have multiple resource bundles.

However, you may need to change a specific string in all places where that string appears without having to change it on a per-instance (page/ADF BC object) basis. For instance, you may need to change "worker" to "employee," "activity" to "task," or "expenditure" to "cost." The method to use is the override resource bundle.

In the Oracle Fusion Middleware Developing Web User Interfaces with Oracle ADF Faces guide, see:

- Internationalizing and Localizing Pages for information about setting up resource bundles.
- How to Define Custom Validator and Converter Messages for All Instances of a Component for general instructions on overriding a resource bundle.

This bundle is specified in the adf-config.xml file and is created at runtime. For Oracle Fusion applications, there are two important points:

- This file must be named FusionAppsOverrideBundle.xlf
- Although an application can have multiple resource bundles, it can have only one override resource bundle.

3.9 Implementing Oracle Enterprise Scheduler Service Workspace and Deployment

Oracle Enterprise Scheduler Service applications, also known as Oracle Enterprise Scheduler Service hosting applications, are deployed to an Oracle Enterprise...
Scheduler Service-configured runtime or cluster that has been pre-deployed with the base ESSAPP infrastructure Java EE application. Following standards, Oracle Enterprise Scheduler Service workspaces exist one per product family and are responsible for containing these supporting projects:

- Oracle Enterprise Scheduler Service projects for containing Job, Job Set, Incompatibility and Schedule Metadata as well as source files for any Java Job implementation.
- ADF data model projects for containing Parameter view object business components.
- ADF user interface projects for containing Parameter task flows.
- Optional servlet or UI task flow projects for development-time testing.
- SuperEss consolidating the Enterprise JavaBeans deployment descriptors for the entire Oracle Enterprise Scheduler Service hosting application. See How to Create the SuperEss Project in the ADF UI Workspace.

**Note:**
All the projects in an Oracle Enterprise Scheduler Service application, regardless of their content type, should have an ADF Business Components Shared Library deployment profile.

A typical Oracle Enterprise Scheduler Service workspace structure resembles Figure 3-5.

*Figure 3-5   Typical Oracle Enterprise Scheduler Service Workspace Structure in JDeveloper*
3.9.1 How to Create the SuperEss Project

In Oracle Fusion Applications, all Oracle Enterprise Scheduler Service workspaces must contain a SuperEss project that contains the EJB deployment descriptors to register the hosted application with the ESSAPP base application, and to register both the MetadataService and RuntimeService EJBs. This technique avoids having multiple projects with conflicting deployment descriptors in a single deployment archive (EAR).

RuntimeService and MetadataService beans are hosted by the ADF UI application and the Oracle Enterprise Scheduler Service hosting application.

If you are creating a new Oracle Enterprise Scheduler Service workspace or your Oracle Enterprise Scheduler Service workspace does not already have a SuperEss project, create one using these steps:

1. In JDeveloper, create the SuperEss project by creating a new Generic Project named SuperEss.
2. In the project properties, create a new EJB-JAR deployment profile.
3. In the File Groups Properties, click New to create a new file group.
4. Name it and set the directory path of both to src/META-INF.
5. In the source directory, create the src/META-INF directory. Create the ejb-jar.xml and weblogic-ejb.jar.xml files. See "Building a Combined Oracle Enterprise Scheduler Application" in the Developing Applications for Oracle Enterprise Scheduler. Save the files in the src/META-INF directory.

After completing these steps, the SuperEss project will be complete. Follow How to Build the EAR/MAR Profiles to build the EAR/MAR deployment profiles. An EAR (Enterprise Archive) is a Java EE archive file that is used in deploying applications on a Java EE application server. Framework applications are deployed using both a generic EAR file, which contains the application and the respective runtime customization, and a targeted EAR file, which contains only the application for deployment to the application server. A MAR (Metadata Archive) is a compressed archive of selected metadata used to deploy metadata content to the MDS Repository.

3.9.2 How to Build the EAR/MAR Profiles

Oracle Enterprise Scheduler Service-hosted applications are built into EAR files and deployed as Java EE applications. The EAR archive must contain the SuperEss EJB JAR, the MAR archive containing all Oracle Enterprise Scheduler Service metadata, and all the Job and Job-related classfiles using JARs in the APP-INF/lib directory. Follow these steps to create the appropriate deployment profiles.

3.9.2.1 Deploying a Project-level Metadata Archive (MAR)

**Note:**

Oracle Enterprise Scheduler Service is used in the instructions because it is the primary, but not only, use case.

To simplify patching of Oracle Enterprise Scheduler Service metadata artifacts and align with code-level patching, it is essential to have project-level deployment artifacts.
To support this requirement, EARs with multiple MAR files can be deployed. This section describes what must be done to properly build Oracle Enterprise Scheduler Service workspaces to support project-level MARs.

3.9.2.1.1 How to Enable Your Application Workspace for Project-level MAR Deployment

In contrast to standard MAR deployment, in which a single .mar file is created as a metadata aggregate from contributors defined from one or more projects in the application workspace, this approach focuses on the creation of a JAR-based deployment profile in each project where the target file is named with a .mar extension. The resultant .mar files are then deployed into the application workspace’s jlib folder, which is added to the top-level directory of the EAR by the EAR deployment profile.

Follow these steps to implement the project-level MAR deployment.

1. Prepare the Application Workspace EAR deployment profile.
   a. Open your Oracle Enterprise Scheduler Service Workspace in JDeveloper.
   b. Open the Application Properties, select the Deployment panel, choose your application’s EAR deployment profile and click Edit.
      These steps will need to be repeated if you have multiple EAR deployments for development or test purposes.
   c. Select File Groups and click New to create a new file group. Leave the type as Packaging and name this group MAR Group.
   d. Leave the remaining values at their defaults and click OK.
   e. Select the Contributors heading beneath the new MAR Group file group and click Add.
   f. Browse to find the application workspace-level jlib directory and click OK.
   g. In the Filters heading beneath the MAR Group, remove all the filters and add these two rules so they display in this order:
      Include *.mar
      Exclude *.*

2. Prepare each metadata project JAR(MAR) deployment profile.
   These steps will need to be repeated if you have multiple projects containing Oracle Enterprise Scheduler Service metadata.
   a. Select the Oracle Enterprise Scheduler Service metadata containing project and open the project properties.
   b. Select the Deployment panel and click New.
   c. If necessary, select the JAR File archive type and provide a meaningful profile name, such as <ProjectName>_MAR.
   d. In the JAR Options panel, change the JAR File destination to point to the application workspace-level jlib directory, and change the file’s .jar extension to .mar.
   e. Select File Groups > Project Output > Contributors and de-select Project Output Directory and Project Dependencies.
3. Complete the Application Workspace EAR deployment profile.
These steps will need to be repeated if you have multiple EAR deployments for test or development purposes.

a. Open the Application Properties.
b. Select the Deployment panel and choose the EAR deployment profile for your application.
c. In the MAR Group file group, select Filters and ensure that all of your project level MAR archives are selected.
d. Select the Application Assembly heading and de-select the application workspace-level MAR profile.
e. Click OK.
f. Select the Profile Dependencies in the left-hand side.
g. Check the boxes for each of the new JAR-based MAR profiles.
h. Click OK.
i. Deploy as EAR and verify.

3.9.2.2 Build the EAR Profile

The EAR profile pulls together all of the previously-created EJB and MAR profiles to build the Oracle Enterprise Scheduler Service-hosted application. All the Oracle Enterprise Scheduler Service workspace projects should have ADF library JAR deployment profiles, and those with Job-supporting implementation classes should be deployed to a directory that can be added to the Oracle Enterprise Scheduler Service EAR’s contributor list.

To create the EAR profile, follow these steps:

1. Open the Application Properties and select the Deployment panel.
2. Click New to create a new deployment, choose EAR File as the profile type, and provide a unique name.
3. In the Application Assembly, choose the SuperESS EJB-JAR profile and nothing else.
4. Select the File Groups menu entry and click New, giving the name APP-INF/lib and assigning the target directory to ess workspace root path/jlib.
5. Under the APP-INF/lib File Group’s contributors, add the directory that holds all of the Job-supporting Implementation classes (not data model projects with ADF Business Components for parameter view objects or parameter task flows).
6. Select the File Groups menu entry and click New, giving the name MAR Group and leave the target directory empty. Under the MAR Group’s contributors, add the directory that holds all the project level mar files (such as Ess/jlib).

7. Click OK.

Note that the EAR profile should contain only the SuperEss EJB JAR, the MAR, and the JAR files for the Job implementation classes. The Oracle Enterprise Scheduler Service hosting application’s EAR file must not contain JARs, descriptors or other artifacts for UI, data model or services functionality. Should your application contain projects with servlet or UI task flows for development testing, they must be bundled into a separate, UI-specific, set of EAR/MAR deployment profiles.

3.9.2.3 Deploy an Oracle Enterprise Scheduler Service Hosting Application

When deploying an Oracle Enterprise Scheduler Service hosting application, the target managed server must have the ESSAPP base application pre-deployed and configured to run against a working Oracle Enterprise Scheduler Service database schema. A database schema is a named collection of objects, such as tables, views, clusters, procedures, packages, attributes, object classes, and their corresponding matching rules, which are associated with a particular user.

For deployment from JDeveloper, you will need to create an Application Server connection in the JDeveloper resources palette before or as part of the deployment activity using the New Connection feature. When your Oracle Enterprise Scheduler Service application is ready for deployment, including all requisite project and application-level profiles, you can initiate deployment by following these steps:

1. Click Deploy > <ear profile name> from the Application menu.

2. Choose Deploy to Application Server and click Next.

3. If no application servers are defined, or the one to which you wish to deploy is not defined, click Add an Application Server. Otherwise, select the server. Do not click Next yet.

4. De-select the Deploy to all server instances in the domain option, because certain libraries needed for deployment of the Oracle Enterprise Scheduler Service hosting application will not be targeted to all the managed servers, and deployment will fail.

5. Click Next.

6. Choose the appropriate managed server and click Next.

7. Click Finish to begin deployment.

JDeveloper will build the EJB JAR and the MAR, and bundle those archives, along with the JAR files, in the APP-INF/lib contributor location. This packaged archive will be sent to the managed server for deployment. During deployment, the Oracle Enterprise Scheduler Service hosting application will register itself through the ESSAPP base application using the ESSAppEndpoint descriptor in your ejb-jar.xml file.

When deployment is finished, jobs can be submitted programmatically or through the Oracle Enterprise Scheduler Service UI submission task flows. These methods are documented in the Developing Applications for Oracle Enterprise Scheduler guide.
3.10 Implementing Oracle Application Development Framework UI Workspace and Projects

Before you can deploy your web project, you need to complete several preliminary steps. These include setting up your web project and configuring your user interface project; creating the SuperESS project; creating the appropriate deployment profiles; and creating and setting up Oracle WebLogic Server.

3.10.1 How to Set Up Your Web Project

When you choose the Oracle Fusion Applications Developer role when starting JDeveloper, many settings default automatically for you. However, there are still certain options that you need to set manually to configure your project.

This section discusses the specific options that need to be set manually to configure your user interface project. The default name for the project that JDeveloper provides is ViewController.

3.10.1.1 Configure Your User Interface Project

This section describes how to configure your user interface project in JDeveloper.

To configure your user interface project:

1. Choose Application Navigator > ViewController project. Right-click and choose Project Properties from the menu.

2. Choose the Project Source Paths category to display the Project Source Paths dialog.

3. In the Default Package field, enter the name of your default package, as shown in Figure 3-6.

Figure 3-6  Project Properties — Project Source Paths Dialog
Many objects are generated automatically and are stored in the default package.

4. Choose the **ADFm Sources** category from the Project Source Paths hierarchy to display the Project Source Paths: ADFm Sources dialog, as shown in Figure 3-7.

**Figure 3-7  Project Properties — Project Source Paths: ADFm Sources Dialog**

The location for all the ADF Metadata sources is the location that is entered in the **ADFm Source Directory** field. You should not have to change the default location.

5. Choose the **Web Application** category from the Project Source Paths hierarchy to display the Project Source Paths: Web Application dialog, as shown in Figure 3-8.
The location for all the HTML content is the location that is entered in the **HTML Root Directory** field. You should not have to change the default location.

6. Choose the **ADF Model** category to display the ADF Model dialog, as shown in Figure 3-9.

The location of the page definition files is based on a combination of the **PageDef sub-package** value, the default package location, and the ADFm Sources directory.

7. Choose the **Deployment** category to display the Deployment dialog.
8. Select **New** to open the Create Deployment Profile dialog, as shown in **Figure 3-10**.

*Figure 3-10*  **Project Properties — Deployment — Create Deployment Profile Dialog**

9. Choose **ADF Library Jar File** from the **Archive Type** list.

   Enter the **Name** as `Adf<projName>` in accordance with the Package Structure and Naming Standards.

10. Click **OK** to save the new deployment profile and close the Create Deployment Profile dialog.

    **Note:**

    The new deployment profile is now listed on the Deployment dialog.

11. Choose the **JSP Tag Libraries** category to display the JSP Tag Libraries dialog, as shown in **Figure 3-11**.
Verify that you have the following tag libraries listed under the **Distributed libraries** folder:

- Applications Core (ViewController) 11.1.1.0.0
- Trinidad HTML Components 1.2

**Note:**

You may have to include additional tag libraries for other features, such as Data Visualization Tools (DVT) and WebCenter. For more information about adding tag libraries to your user interface project, see [Adding the Applications Core Tag Library to Your User Interface Project.](#)

12. Choose the **Libraries and Classpath** category to display the Libraries and Classpath dialog, as shown in Figure 3-12.
Verify that the libraries listed in Figure 3-12 have been attached to your user interface project. As with tag libraries, you may have to add additional libraries.

**Important:** The Applications Core library *must* be included to make the page's design tab visual, instead of showing nesting boxes of XML elements. To add the Applications Core library:

- Click Add Library.
- In the Add Library list, select the Applications Core entry, as shown in Figure 3-13 and click **OK**.

- Verify that the Applications Core library now is included in the Libraries and Classpath list, as shown in Figure 3-14.
13. Choose the Resource Bundle category to display the Resource Bundle dialog, as shown in Figure 3-15.

**Figure 3-15  Project Properties — Resource Bundle Dialog**

Verify that the Resource Bundle Type is set to Xliff.

**Note:**

The default settings will be correct when you start JDeveloper using the Oracle Fusion Applications Developer role.

14. Choose the Technology Scope category to display the Technology Scope dialog, as shown in Figure 3-16.
Verify that the technology scopes that are selected in Figure 3-16 are selected for your project.

Note:
This selection is limited to what is available, by default, in the New Gallery. To see other types of objects, choose All technologies from the New Gallery.

3.10.2 How to Create the SuperEss Project in the ADF UI Workspace

Follow the steps in How to Create the SuperEss Project. The differences are that the ejb-jar.xml file will have no ESSAppEndpoint MDB and the weblogic-ejb.jar.xml file will be empty.

3.10.3 How to Deploy Your Web Project

Deployment is the process of packaging application files and artifacts and transferring them to a target application server to be run. During application development using JDeveloper, developers can test the application using Integrated WebLogic Server that is built into the JDeveloper installation, or they can use JDeveloper to directly deploy to a standalone application server.

After the application has been developed, administrators can deploy it to production application servers.
Note:

This section assumes that you are deploying a web project to Standalone WebLogic Server. Creating deployment profiles is not necessary if you are running the project in Integrated WebLogic Server from within JDeveloper. In this case, JDeveloper, behind the scenes, creates an in-memory deployment profile.

For other deployment options, see Deployment Techniques for Development or Production Environments in the Developing Fusion Web Applications with Oracle Application Development Framework.

To deploy the web project to Standalone Weblogic Server, you must:

- Create a Web Archive (WAR) deployment profile. This file is used in deploying applications on a Java EE application server. WAR files encapsulate in a single module all of the components necessary to run an application. WAR files typically contain an application’s servlet, JSP, and JSF JSP components. To create the WAR deployment profile, see How to Create Deployment Profiles in the Developing Fusion Web Applications with Oracle Application Development Framework.

- When you have defined the WAR, create an EAR deployment profile that includes the WAR profile, for the application. To create an EAR deployment profile, see Creating an Application-Level EAR Deployment Profile in the Developing Fusion Web Applications with Oracle Application Development Framework.

- Create, if necessary, and prepare the standalone application server for deployment. To run ADF applications, you must install the standalone application server with the ADF runtime. You can include the ADF runtime during a new application server installation or you can install the ADF runtime into an existing application server installation.

- Deploy the application using one of these methods:
  - Oracle Enterprise Manager Fusion Middleware Control
  - WebLogic Scripting Tool (WLST) commands (see Deployment Commands in the WLST Command Reference for WebLogic Server) or WebSphere Application Server (wsadmin) commands
  - Command scripts and Ant scripts
  - Oracle WebLogic Administration Console or IBM WebSphere Administrative Console
Get Started with Business Services

This chapter provides an overview of ADF Business Components, validators, list of values (LOVs), and data types. It also discusses migrating PL/SQL to Java, batch processing, and extensibility and reusability. Also included is an overview of services.

This chapter includes the following sections:

• Introduction to Implementing Business Logic
• Understanding Validators
• Understanding List of Values (LOV)
• Understanding Batch Processing
• Understanding Extensibility and Reusability
• Understanding Services
• About Using the Declarative Approach

4.1 Introduction to Implementing Business Logic

The core business logic is implemented in one or more business components that are provided in ADF Business Components. Entity objects, view objects, and application modules are the key business components that are discussed in this section.

4.1.1 About Entity Objects

An entity object represents a row in a database table. It encapsulates the business logic and database storage details of your business entities. It simplifies modifying its data by handling Data Manipulation Language (DML) operations automatically. There are two general classifications of business logic that are placed on the entity object:

• Standard business and validation logic
• Specialized business functions

4.1.1.1 Standard Business and Validation Logic

An entity has a life cycle; customized business rules can be added to an entity object at various places to be executed in different phases of its life cycle.

The entity object should contain all logic that is invoked during entity object life cycle events. This comprises logic for create, initDefaults, all validation (including attribute validation, entity validation and cross entity validation), DML, and so on. In other words, the entity object encapsulates the rules that ensure the entity object is created and remains in a valid state.
Note:
All logic existing in one entity object Java class not required. It's valid for the entity object to call utility classes for code modularity purposes.

If a business rule can be defined declaratively, you should always use the declarative approach. For example, if an attribute has a constant default value, then you should specify it in the Entity Object wizard rather than coding it. You should also first consider using declarative validators for your validation logic, which is explained in Understanding Validators.

The life cycle of an entity object begins with being created as a new entity object or fetched from the database as an unmodified entity object. The entity object can then be modified or removed. Only new, modified, or removed entity objects are in the transaction pending change list and are posted to the database when the transaction is committed.

For more information about the key events in the entity objects life cycle and where you can add entity object business logic programmatically, see the Introduction to Programmatic Business Rules section in the "Implementing Validation and Business Rules Programmatically" chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

4.1.1.2 Specialized Business Functions
The entity object is the core business object that is used to encapsulate task-level business logic. It is shared by both user interfaces (UIs) and services. Core business functions and tasks should be placed on the entity object as custom methods for maximum reusability. Corresponding methods on the view object and application modules should delegate to these functions on the entity object. Examples of business functions are approvePurchaseOrder and hireApplicant. Internally, these custom methods can be implemented using Java or may invoke legacy PL/SQL.

Custom business functions that are not invoked during entity object life cycle events should be placed in either the entity object or model application module. Generally these are the custom business functions that are required by the UI and Service application module.

4.1.2 About View Objects
A view object represents a SQL query and also collaborates with entity objects to consistently validate and save the changes when end users modify data in the UI. The relationships between view objects are captured using view links. View objects are used to present your business data for the specific needs of a given application scenario or task, and generally don't contain business logic.

However, view objects may have additional attributes that do not exist in the underlying entity objects, which are used to store some calculated values. Usually you define different view objects for supporting services and UIs:

- Service view object: Represents an outfacing business object and contains only the attributes in the business object. For example, it contains the foreign key ID attribute such as SupplierID, but it does not contain foreign key reference attributes such as SupplierName.
• **UI view object**: This view object may contain additional UI flags and calculated attributes that are used for a particular UI. In addition, the UI view object may join to other tables for additional foreign key attribute references.

**Note:**
A service view object must be versioned to support service versioning. However, there is no versioning requirement for an internal UI view object.

### 4.1.3 About Application Modules

An application module encapsulates an active data model and the business functions for a logical unit of work related to an end-user task. The active data model is defined as a collection of view object instances.

The methods on the application module are used to encapsulate task-level business logic, although these methods should delegate to the methods on entity objects whenever possible. If you have an option to put your business logic either on an entity object or an application module, then you should always put it on the entity object. This is because the entity object owns the business object and also for better reusability. The task-level validations that span multiple related parent-child entities, such as purchase order header and lines, should be put on the parent entity. It is also important that the entity object should not trust the incoming data and always perform all validations.

Application modules can be used to support UIs or define services. Usually you want to have two separate application modules for the two different purposes because:

• **UI application modules** may contain additional view objects and context values that are only required for a particular UI, but not needed by a business service.

• **Application modules** that define public services are versioned, but internal UI application modules are not.

UI application modules and service application modules share the underlying entity objects as shown in Figure 4-1. A UI application module can also call a service.

![UI Application Module and Service Application Module](image.png)

#### 4.2 Understanding Validators

In Fusion, you should use validators to implement the validation logic. Validators are added declaratively, which provides visibility and personalizability to customers as well as the benefit of being easy to use and maintain. Validation view objects can be attached to entity objects declaratively as view accessors, which can then be used in declarative validators.
For more information about how to use validators and Groovy (a Java-like scripting language), see Defining Defaulting and Derivation Logic.

### 4.3 Understanding List of Values (LOV)

List of Values (LOV) is the mechanism to specify a list of valid values for an attribute in a view object. There are basically two parts involved when a LOV is defined: the base object and the LOV object. The base object is a view object, which contains the attribute whose list of valid values need to be defined, such as a PurchaseOrder view object containing a BuyerId attribute. The LOV object is a normal view object that contains the list of valid values, such as a Buyer view object containing all the valid buyers. You should use the view object design time wizard to add a List Value on the BuyerId attribute to associate with the Buyer view object.

In Fusion, the LOV metadata is defined on the server using ADF Business Components, and this drives and defaults the UI controls to automatically render the LOV bound items accordingly when you define a page, such as LOV and poplist controls.

It's important to define the LOV and entity validators to share the same view object instance to avoid redundant database round-trips for validation. To achieve this, you should add the LOV view object as a view accessor in the entity object, and use the view accessor to define entity level validators. The same view accessor is available at the view object level and is needed to define the LOV. When the user picks up a row from LOV on the UI, the row is placed on the LOV view object cache. The entity object validation then hits the cache instead of going against the database. A LOV/validation view object can also be defined as a global data source that is shared among all the users.

For more information, see the “Sharing Application Module View Instances” chapter, in the Developing Fusion Web Applications with Oracle Application Development Framework.

### 4.4 Understanding Batch Processing

Depending on the use case, different approaches should be considered to achieve the best performance.

For the use case of complicated bulk processing, such as very high volume or multiple step processing, a combination of multiple techniques must considered. For example, ADF Business Components, C, PL/SQL, SOA, Oracle Enterprise Scheduler (ESS), and so on.

For the other use cases, such as integration with third parties or data migration, if the data volume is low to medium, then ADF Business Components service is necessary. Internally, a combination of interface table and PL/SQL can be used to handle large amount of data, complicated processing, and validation logic. The inbound data is loaded into an interface table through a service and then the PL/SQL API is executed to process the data.

---

**Note:**

You should still provide services for all objects, and double code the high performance alternatives when necessary.
4.5 Understanding Extensibility and Reusability

Fusion web applications are extensible applications, which can be tailored to fit the business practices specific to a customer, locale or an industry. Adaptation through Business Editor enables you to extend Oracle applications declaratively, which satisfies most of the extensibility requirements. You can also use the programmatic extensibility feature to address additional use cases.

4.6 Understanding Services

A service is a set of operations defined by an interface that can be used by other components. In Fusion, applications use both ADF Business Components services and SOA services. ADF Business Components services should be created to manage business objects and SOA services are for orchestration and business processes.

In Fusion, you make your data and business logic available via UIs and services. For more information about services, see Developing Services.

4.7 About Using the Declarative Approach

When building your model objects, you should use the declarative approach whenever possible. For example, when defining your view objects, use declarative SQL mode whenever possible, base your view objects on entity objects, and use view criteria.

4.7.1 About Defining View Objects Using the Declarative Approach

When you define your view objects, use declarative SQL mode wherever possible. The next option is to use normal SQL mode.

When building your view objects, use declarative SQL mode wherever possible. Reasons for not using declarative SQL include:

- You have a complicated query and the WHERE clause cannot be implemented using view criteria.
- Your query includes derived attributes that cannot be implemented as calculated attributes based on a SQL expression.

If you are unable to use declarative SQL mode, you should try and use normal SQL mode, which gives you full control over the WHERE clause. Only use expert mode if other modes do not work. However, you should still base the view object on an entity object when the query supports it.

Non-expert mode view objects are metadata based and more declarative instead of SQL based. The declarative approach gives you benefits such as:

- Increased development productivity:
  - Proven experience from PeopleSoft and Siebel.
  - Removes the requirement for you to tune each view object.
- Easier to perform dependency and impact analysis.
- Can be extended more robustly.
Declarative SQL mode is recommended because it is an even more declarative approach to defining the view object than normal mode.

- The runtime query optimization feature is enabled only when you use declarative SQL mode. ADF Business Components makes runtime changes to the SQL based on usage such as column pruning to improve performance.
- Declarative SQL optimization means you can consider creating view objects that can be reused in multiple UIs without impacting runtime performance.

For more information about how to set the SQL mode, see the Working with View Objects in Declarative SQL Mode section of the "Defining SQL Queries Using View Objects" chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

4.7.1.1 About Using Entity Object Based View Objects

All view objects (including read-only view objects) should be based on entity objects unless:

- Entity object based view objects are not supported for the SQL statement you are using, for example:
  - Union
  - Select Distinct
  - Group By
- Your data doesn’t come from the database. For example, the data comes from external files.

Even if you have to use expert mode view object, you should still base your view object on top of entity objects because:

- The attributes from the entity object are still declaratively defined so that you can partially benefit from the declarative approach.
- Multiple view objects based on the same entity object can automatically see updates made by any one of them.
- Updated reference information is reflected when foreign key attribute values are changed.
- Metadata, (UI hints, associations, and other attributes), are automatically propagated up to the view objects from entity objects.
- New row management, such as view link consistency, only works with an entity object-based view object.
- findByKey doesn’t work for view objects with no entity usage unless you turn on the key management at the view object level (and this will add significant resources and CPU time). The findByKey method is a frequently invoked by any operation that involves setting the current row, such as clicking a row on an ADF Faces rich client table:
  - findByKey does not find the matching view row in the view object cache if the key management is not enabled.
findByKey adds the row fetched from the database into view object cache even if the view object already has the same row in cache if the key management is not enabled.

- Updatable view objects must be based on entity objects so that view objects can coordinate with the underlying entity objects to perform DML.

4.7.1.2 About Utilizing View Criteria

Instead of directly setting the WHERE clause, use declarative named view criteria whenever possible. Named criteria can be re-used in the UI and in the service interface. Also, it supports customization better and is required for declarative SQL mode.

In parallel, always use named bind parameters. Define the named bind parameters during design time if possible. Otherwise, add the named bind parameters programmatically. Named bind parameters are much easier to understand and manage than the indexed bind parameters so therefore, the code is easier to develop and maintain. If the same bind parameter appear multiple times in the WHERE clause, you only need to bind it once.
Service-oriented development is based on the concept of services. It is the realization of business functionality via software that customers can use to compose new business applications by using existing services in the context of new or modified business processes. This chapter describes how you should design and develop the services to make them useful for both Oracle Fusion Applications and for customers. It also covers how the services are consumed.

This chapter contains the following sections:

- Introduction to Services
- Designing the Service Interface
- About Developing Services
- About Invoking Services

### 5.1 Introduction to Services

A service is defined in terms of its interface, which is the way the service is exposed to the outside world. The service interface includes a set of operations the service supports, a set of parameters (defining data required for interaction with the service), and communication protocol used for data transfer and actual service invocation. Grouping of the methods in the service interface is defined by business functionality of the service.

Services have the following characteristics:

- Business-driven
- Coarse-grained
- Process-centric
- Stateless invocation
- Loosely coupled
- Distributed
- Standards-based

In Oracle Fusion, applications use both ADF Business Components services and service-oriented architecture (SOA) services. ADF Business Components services should be created to manage business objects and SOA services are for orchestration and business processes. SOA services use business object services to encapsulate business processes as illustrated in Figure 5-1:
This chapter focuses on business object services that are implemented using ADF Business Components services.

In Oracle Fusion, you make business objects and related business logic available via user interfaces (UIs) and services. A single general-purpose service can satisfy multiple use cases such as:

- Programmatic application programming interface (API) calls required by external customers, cross-pillar integration, or third-party integration.
- Business Process Execution Language (BPEL) process flows and composite applications
- Business-to-business (B2B) integration through standardized documents
- Foreign UI technologies such as Microsoft .NET framework
- XML-based reporting
- Desktop applications such as Excel and Access

5.2 Designing the Service Interface

A service is a public interface, and it requires careful design to make it useful. For ADF Business Components services, on which this chapter focuses, the purpose of these services is managing business objects, including the generic lifecycle of the objects (create/update/delete/query), and special actions that can be applied to the objects.

Therefore, designing these services will start from identifying the business objects, continue with identifying service operations on business objects, and grouping the operations into services. Exceptions, warnings, or informational messages need to be defined for each operation and service as part of the service interface design process.

5.2.1 About Identifying Business Objects

Identify which business objects that you want to expose from the service interface.

5.2.1.1 About Business Object Attributes

The shape of the business object is very important. You need to include all the attributes that represent the objects, but exclude anything that is of no business value.

The business object must contain the following attributes:
• Primary key attributes, including the system generated surrogate keys when defined.
• Attributes that are of business value to the consumer. This will include most attributes of the physical tables.

5.2.2 Identifying Service Operations on the Business Objects

The service operations are the actions that can be performed on business objects, such as create and delete.

5.2.2.1 Types of Operations

Standard operations and Custom operations are the two types of service operations that are supported by ADF Business Components service.

Standard Operations

The primary purpose of standard service operations is to locate a business object and handle its persistence. This includes storage, manipulation and retrieval of data, locking, transaction management, business rule validation, and bulk processing. ADF Business Components auto-generates the following groups of standard service operations:

• CRUD (Create, Read, Update, Delete) Operations:
  - get<businessObjectName>: get a single business object by primary key.
  - create<businessObjectName>: create a single business object.
  - update<businessObjectName>: update a single business object.
  - delete<businessObjectName>: delete a single business object by primary key.
  - merge<businessObjectName>: update a business object if exists, otherwise create new one.

• Find Operation:
  - find<businessObjectName>: find and return a list of business objects by find criteria.

• Bulk Processing Operations:
  - process<businessObjectName>: process a list of business objects via a CRUD command.
  - processCS<businessObjectName>: process a list of business objects via a change summary.

• Control Hints Operations:
  - List<AttrCtrlHints>
  - getDfltCtrlHints(String viewName, String localeName): takes the view object name and a locale and returns the base UI hints for that locale.
Custom Operations

Custom service operations encapsulate complex business rules and may coordinate execution of two or more data-centric operations within one atomic transaction.

5.2.2.2 About Business-Object Operations

All core business functions should be exposed in services. When developing a list of business object operations, consider the entire application life cycle from creation to deletion. Also, consider potential use cases that are required by others. Typically, the functionality exposed from UI should be achievable from services. For example, the following list includes many of the operations associated with requisitions and purchase orders:

- **Operations associated with Requisitions:**
  - create
  - merge
  - update
  - delete requisition
  - delete requisition line
  - delete requisition distribution
  - copy requisition
  - get requisition
  - cancel
  - approve (including all approval states such as approve, reject, and pre-approve)
  - view approval history

- **Operations associated with Purchase Order:**
  - create
  - delete purchase order
  - delete purchase order line
  - delete purchase order shipment
  - delete purchase order distribution
  - copy purchase order
  - merge
  - update (change)
  - acknowledge
  - get purchase order
  - cancel purchase order
- cancel purchase order line
- approve (including all approval states)
- view approval history
- close

Typically, you should include all the standard operations, although the delete operation should only be included if supported by the business object.

Custom operations must be coarse-grained and must reflect a business task to be carried out. activateCustomerParty, suspendOrder, and closeServiceRequest are some examples.

### 5.2.2.3 General Guidelines for Defining Service Operations

There are general guidelines you must follow when defining service operations.

**Be generic where it makes sense**

Since most Oracle Fusion services serve multiple use cases as listed in *About Business-Object Operations*, services should not be designed narrowly for only one use case at the exclusion of others. Instead, services should be designed from the start to be general purpose and contain APIs that can serve the widest use cases. This is especially important to consider for common business functions that are initially required for the UI or to meet enhancement requests from other products. It should be the conceptual essence of the use case that drives the interface, not the fine-grained specifics of one consumer. A consumer can be seen as a representative of a specific use case, but the provider should always apply well-measured foresight when defining the interface details. Creating general purpose APIs from the beginning will:

- Prevent method explosion as other similar use cases are requested.
- Increase reuse by multiple use case.

For example:

There is a requirement to return a person’s name based on `personId`. To satisfy that requirement, you might want to create an operation such as:

```java
public String getPersonName(Long personId)
```

However, it is likely that this service will soon be adopted by more consumers, which will require other attributes of a person. In this example, it makes sense to return the person object instead of just the name into the first specification of the service interface, such as:

```java
public Person getPerson(Long personId)
```

**Leverage standards wherever possible**

When there is an existing industry standard that can be applied to your service, leveraging the standard is definitely recommended to avoid costly negotiation of proprietary interfaces.

However, in many situations either no standard exists or the standard does not optimally support your business need. In this case you should make the interface as generic as possible for your given group of consumers. This will make sure that the interface stays stable as more consumers adopt it, while being highly useful for your given business processes. With good strategic planning it is possible to define generic interfaces for a defined subset of stakeholders.
Service operation granularity

Because it is possible to call services across a network, the service operations should be generally coarse-grained. That is, a service operation should wrap a substantial body of application logic, delivering value that justifies the latency cost of a network request. Similarly, services should expose coarse-grained operations. Rather than expose many operations that each manipulate small amounts of state, services should expose fewer operations that allow a single request to perform a complete function.

Compensating service operations

For operations that involve data manipulation, a clear strategy for compensation must be defined. Services are frequently distributed remotely and there is no central transaction coordinator with sufficient control over all resources. This is inherent in Simple Object Access Protocol (SOAP), which is predominantly used in the web services space and therefore, a two-phase commit protocol cannot be enforced. Also, two-phase commit implies resource locking, which may lead to scalability and availability issues if locks are held for longer periods.

To allow for service operations to be undone, in certain business scenarios it may be possible to offer compensating service operations. These operations are used to revert the system back to the state before the original operation was invoked. Providers and consumers must agree on the conditions under which an original operation can be undone and what information is required to achieve the compensating effect.

In most cases, the decision to provide a compensating operation is primarily functional. It might technically be possible to delete an existing purchase order, but functionally it is only correct to cancel it after it has been submitted for approval. Not all operations should, by default, be paired with a compensating operation. Compensating operations should be provided only if the business process demands that the system can be rolled back into the original state.

If it is not possible to provide a compensating operation, you should strive to make the service operations truly accomplishable (idempotent). Making the operations idempotent means that the operation can be invoked multiple times with the same payload but still the result will be the same with no undesirable effects. If neither providing compensating service operations nor making the service operations idempotent is feasible, processes should be designed in such a way that the service operation would never be called more than once under any circumstances for a specific request.

Service operation parameters

Each service operation can have zero or more parameters. Each parameter can be a primitive type (String, Date, and so on), a complex type represented as a Service Data Object (SDO), or a List of a primitive type or SDO. Complex types can in turn contain nested complex types.

- Long Parameter List or Complex Types?

You should consider using complex types in a service operation instead of using a long list of individual parameters unless the parameter list can be reduced to a short list of simple types (3-5).

For example:

A service operation `updatePerson()` takes a compound complex type of `Person`, which includes several individual attributes such as `BirthDate`, a collection of `PersonName`, and a collection of `PersonAddress`, and so on. The reasons are:
- Taking a list of individual parameters leads to a not so clean operation signature:

- Adding an optional attribute on a complex type doesn’t break compatibility, but adding a new parameter in a method does.

- In the `updatePerson` example, if the person’s email address must be updated, the operation that takes a `Person` can stay unchanged.

- **Complex Types or Primary Keys?**

  As an alternative to complex types, business object Primary Keys can be used in operation signatures in certain cases.

  Auto-generated data-centric standard operations, such as create, update, delete, merge, and Bulk Processing take complex business object types as parameters. Auto-generated `get()` takes primary keys.

  Custom methods may take business object primary keys or developer key as parameters, when the key attribute is used to look up the business object, such as `terminateEmployee`. Complex business object documents should be passed primarily to the data-centric custom methods: `validatePerson()`, `promoteEmployee()`, `formatPersonAddress()`, and so on.

**Example 5-1  Service Operation on a List of Parameters**

```java
void updatePerson (Date BirthDate, PersonName[] Names, PersonAddress[] Addresses);
```

### 5.2.3 How to Identify Services

A service is a grouping of operations. Often this grouping is by the business object it maintains, which is especially true for the CRUD operations. In most cases, one service per business object provides a more manageable hierarchy. For example, the business object `Person` could be offering all operations that can be performed on it as a service called `PersonService`.

After you have identified your services and what business object(s) they include, the list(s) of the corresponding operations that were identified in About Business-Object Operations provide the list of candidate methods for each service.

Services from other products that may compliment this list are not included. For example, a `SupplierService` and `InvoiceService` provide detailed information about suppliers and invoices respectively. The procurement services should identify only who the supplier is in various transactions and provide information on procurement-specific supplier data such as, supplier price, quality and on-time delivery performance. It should not provide core supplier operations like creating, updating, deleting, and so on because that is the responsibility of the `SupplierService`.

It’s important that the Oracle Fusion services compliment one other. Therefore, after you’ve identified your working list of services, coordinate with related products to ensure that you have not duplicated efforts or created confusing and conflicting APIs. Also, communicate any expectations that you have of their services.

### 5.2.4 About Defining Service Exceptions and Information

Services might throw exceptions or warnings when there is issue of processing inbound request. Sometimes it is desirable to return informational messages along with the response. It is an important step to define when to return exceptions,
warnings, and informational messages, and what messages should be returned. This is especially true for the bulk processing operations with partial failure.

5.2.4.1 Defining Service Exceptions

After the required criteria for successful execution of a service operation is agreed upon, all stakeholders must then define a complete set of error conditions. You must define which exceptions can happen and which information should be reported back to the consumer due to an exception. Exception processing should be consistently implemented across all operations in the application. If one operation throws an exception while another returns an empty collection, the consumers perceive the services as unstable and unpredictable. The reported exception should contain as much information as possible so that the consumers can pinpoint the problem easily.

All service operations are delegated to the underlying ADF Business Components objects and their methods. As a service provider, you just need to implement your validation logic and business rules in your server side objects, define appropriate error messages declaratively, or throw appropriate JboExceptions programmatically.

Oracle ADF has one generic exception or fault to handle all ADF Business Components exceptions. Whenever an exception is thrown from the underlying ADF Business Components object for one of the service standard or custom methods, the exception is thrown as a Service Exception, which contains all of the information available from the original thrown exception. This also includes support for bundled exceptions.

5.2.4.2 Defining Partial Failure and Bulk Processing

Services can support partial failure during bulk processing of data, which can be very useful. For example, if the client loads a large amount of data using batch load applications, the occurrence of one or more failures does not prevent the continued posting of other unrelated data.

During design of your services, you need to decide whether partial failure should be enabled for a business object including details. For example, a purchase order business object includes a header, lines for each header, and shipments for each line. The partial failure switch is set on each level including header, line, and shipment. Usually the top level object should allow partial failure, but the decision on the detail level depends on whether it make sense to simply skip that object if it fails. You must ask yourself the question: “Does it make sense to still post the other lines and the header if one line fails?” In some cases, you may need to preserve the integrity of the business object and not allow the object to be posted with partially populated children.

Caution:

The partial failure mode is only used in the processXXX API and this API also uses a runtime partial failure flag in the ProcessControl parameter. This means the partial failure feature is only enabled when both the design time flag and the runtime flag are enabled.

5.2.4.3 Defining Informational Messages

Informational messages are not exceptions and won’t affect the current transaction. However, these messages may be useful to the clients. For example, you may want to know when the system automatically transfers money from your saving account to your checking account because there may not enough funds in your checking account when your check is cashed out.
The service provider needs to define a complete list of informational messages as well as the conditions that these messages should be returned.

5.3 About Developing Services

After you design the service interface, you now must implement those services.

Note:
This is from the service provider perspective.

For more information see “Implementing Business Services with Application Modules” in Developing Fusion Web Applications with Oracle Application Development Framework.

5.3.1 About Creating Service Data Objects

In Oracle Fusion, service data objects (SDOs) are used to expose business objects in services. SDO is an industry standard, and it provides a unifying API and architecture that allows SOA applications handle data from heterogeneous sources, including relational databases, Web services, and enterprise information systems. Oracle has implemented it based on the SDO specification.

Each SDO must be backed by a view object. Hence, you don’t create the standalone SDO. Instead, you always start with a view object, and service-enable the view object using the "Generate Service Data Object Class" option in the view object wizard’s Java panel to create SDO.

This implies that you might need to create transient view object when there is no corresponding entity object. For example, you want to use a SDO to capture parameters when number of parameters is big.

5.3.1.1 SDO Attributes

When you create the view object, make sure the attributes are properly defined. See About Business Object Attributes for information about what attributes should be included in SDO. You can hide view object attribute from SDO if you need to by navigating to the attribute editor and de-selecting the SDO Property checkbox.

Note:
If a view object extends another one, then the subtype SDO cannot hide or unhide an attribute that is defined in the base SDO.

The SDO attribute name is derived from the view object attribute name, and you cannot override it.

5.3.1.1.1 Ensure Attribute Type Consistency

In general, the SDO attribute type is defaulted from the view object attribute, and you can just use the default value. However, the following types are exceptions:

- Boolean type
- AmountType
• MeasureType

**Boolean Type:**

A SDO attribute in boolean domain (only have true/false value) must be of Boolean type.

Oracle ADF provides a feature that maps between the view object attribute String type and the SDO attribute Boolean type. This feature will allow you to continue using String in the view object attribute, but use Boolean in SDO.

To expose a Boolean domain view object attribute of String type as Boolean type in SDO, do the following:

1. Determine the possible values of the attribute. Since it is of Boolean domain, there are only two possible values: **Y/N, T/F, or 1/0**.

2. If needed, create a property set. This property set is used to indicate how Oracle ADF should convert between String and Boolean. Oracle ADF provides property sets for Y/N or T/F or 1/0. If the possible values are not Y/N or T/F or 1/0, then you will need to create a property set. Navigate to the **New Object Gallery**, and select **Property Sets** under **ADF Business Components**.

3. Navigate to the Source view and add the following element as a child of the Domain element:

   ```xml
   <Domain>
     ....
     <Properties>
       <SchemaBasedProperties>
         <BooleanValueMapping>
           <ValueMapping JavaStringValue="true" StorageValue="A"/>
           <ValueMapping JavaStringValue="false" StorageValue="I"/>
         </BooleanValueMapping>
       </SchemaBasedProperties>
     </Properties>
   </Domain>
   
   This example assumes that the possible values are "A" and "T", where "A" means true and "T" means false. You should replace these values with your own.

4. Specify the property set for the attribute. If the view object attribute is entity-object based, then go to the entity object. Otherwise, go to the view object. Navigate to the Source view and, in the corresponding Attribute or ViewAttribute section, add the **TypeValueMapPropertySet** attribute. For example:

   ```xml
   <Attribute
     Name="_Flag"
     ....
     TypeValueMapPropertySet="oracle.jbo.valuemaps.BooleanYNPropertySet">
   ....
   </Attribute>
   
   If your attribute is of Y/N domain, then use
   oracle.jbo.valuemaps.BooleanYNPropertySet as the property set. If it is of T/F domain, then use oracle.jbo.valuemaps.BooleanTFPropertySet. If it is of 1/0 domain, then use oracle.jbo.valuemaps.Boolean10PropertySet. Otherwise, you will need to use your own property set, which you created in the previous step.

5. Navigate to **View Object wizard > Java > Generate Service Data Object Class** and generate or regenerate the SDO.
AmountType:

An SDO attribute storing a price or amount could use the AmountType datatype. Examples are 100 USD or 35.3 RMB. A number and a currency code are always present in this type of attribute. AmountType is a complex type and defined as:

```xml
<xsd:complexType name="AmountType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:decimal">
      <xsd:attribute name="currencyCode" type="xsd:normalizedString" use="optional"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

Therefore, a SalePrice attribute of 100 USD will be represented in xml as:

```xml
<SalePrice currencyCode="USD">100</SalePrice>
```

To use the AmountType, two attributes in view objects must be present: One for the number/amount itself, and the other for the currency code. To use the AmountType, do the following:

1. Navigate to View Object wizard >Attributes and select the specified Price or Amount attribute. Then, click Edit.

2. In the View Attribute panel, select AmountType as the XSD Type.

3. In the corresponding CurrencyCode field, choose the attribute that is used to determine the currency code. If the currency is not defined in the current view object and derived from another place, first create a transient attribute and add logic to populate the transient attribute value, for example, via a Groovy expression. For the transient currency code, override the setter method of the ViewRowImpl class, and validate that the new value is the same as the source attribute. For example, if you have a currency code defined at the parent level, then the transient currency code attribute at the child level should be always the same as the parent level currency code value.

4. Keep the currency code in the SDO payload. In another words, always expose the currency code attribute in SDO, and don’t disable "SDO Property". This will allow the consumer to change the currency code even if the price/amount is not present.

   During runtime, if the payload contains conflicting currency codes, then an exception will be thrown from Oracle ADF. For example, an exception will be thrown if the service payload is something like the following:

   ```xml
   <SalePrice currencyCode="USD">100</SalePrice>
   <ListPrice currencyCode="RMB">100</SalePrice>
   ```

   Instead, both SalePrice and ListPrice should contain same value for the currencyCode attribute:

   ```xml
   <SalePrice currencyCode="USD">100</SalePrice>
   <ListPrice currencyCode="USD">100</SalePrice>
   ```

   Or the currencyCode is specified in one element and omitted from the other:

   ```xml
   <SalePrice currencyCode="USD">100</SalePrice>
   <ListPrice>100</ListPrice>
   ```

MeasureType:
An SDO attribute storing a quantity could use the MeasureType data type. Examples of quantities are: 10 meters or 105.3 pounds. Two things are always associated with a quantity: A number and a unit of measure. MeasureType is a complex type and defined as the following:

```xml
<xsd:complexType name="MeasureType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:decimal">
      <xsd:attribute name="unitCode" type="xsd:normalizedString" use="optional"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

Therefore, a Length attribute of 10 meters will be represented in XML as:

```xml
<Length unitCode="meter">10</Length>
```

In a view object, there are two attributes related to this one quantity: The number attribute and the unit of measure attribute. To define the attribute as MeasureType, do the following:

1. Navigate to **View Object wizard** > **Attributes** and select the specified Quantity attribute. Then, click **Edit**.
2. In the **View Attribute** panel, select MeasureType as the XSD Type.
3. In the corresponding unitCode field, choose the attribute that is used to determine the unit of measure. If the unit of measure is derived from another place, create a transient attribute and add logic to populate the transient attribute value, for example, via a Groovy expression. For the transient currency code, override the setter method of the **ViewRowImpl** class, and validate that the new value is the same as the source attribute.
4. Keep the unit of measure attribute in the SDO. In another words, always expose the unit of measure attribute in SDO, and don't disable "SDO Property". This will allow consumers to change the unit of measure even if the quantity itself is not present.

As with AmountType, an exception is thrown by Oracle ADF if the payload contains conflicting unit codes.

### 5.3.1.2 Parent-Child Relationships

For parent-child relationships, you should define two view objects, one for the parent and one for the child, and then define a view link between them. You must also have the destination accessor generated so that the service framework is unable to query or post the child along with the parent.

For composite object, you should create a composite association between the parent entity object and the child entity object, and base the view link on the association. However, in cases where composite associations cannot be defined you must add a custom property, `SERVICE_PROCESS_CHILDREN=true`, to the entity association or view link. This allows for the child objects to be processed along with the parent object (in `createXXX`, `updateXXX`, `mergeXXX`, `deleteXXX`, and `processXXX`). Reasons for cases where composite associations cannot be defined include:

- A child has multiple parents but the relationship is really composite.

When there is an entity association and the association has the destination accessor generated, then you should add the custom property in the association. When an
association doesn’t exist such as flexfield or the association doesn’t have the destination accessor generated, then you must add the same property to the view link.

5.3.1.3 About Enabling Partial Failure

The default setting for partial failure is not enabled. To enable partial failure, add the PARTIAL_FAILURE_ALLOWED custom property on the view object and set the value to true.

To determine if you should enable partial failure, see Defining Partial Failure and Bulk Processing.

5.3.1.4 About Enabling Support Warnings

There is a design time flag to indicate whether the informational messages are enabled or not for each view object and service data object. The default setting for this flag is off, and you need to go to the view object editor’s Java tab and select the Support Warnings field.

---

**Note:**

Signatures of the service operations that ADF Business Components generate vary depending on this Support Warnings flag. If you change this flag in a future release, your service will no longer be backward compatible. In addition, when partial failure is on, the exceptions are not thrown from the service invocation. Instead, the exceptions are reported as warnings, and the caller can only receive these warnings if the Support Warnings flag on the service view object is turned on. Therefore, you must turn on the Support Warnings flag for the top-level service data objects that are exposed directly in the process methods.

---

For the purchase order header, line, and shipment example, if your service includes the processPurchaseOrders API that takes a list of purchase order headers, then you must enable Support Warnings in the purchase order header view object. If your service also includes the processLines API that takes a list of purchase order lines, then you also need to enable Support Warnings in the line view object. For the other detail level service data objects, you should take a more proactive approach and define your business object as supporting informational messages if you think you will need this feature in the future.

5.3.1.5 Defining a List of Values (LOV) to Resolve Foreign Key ID

Except the services that are only consumable only within Fusion Applications, not only the foreign key ID attributes but also the foreign alternate key attributes should be added to the SDO, when the foreign key ID is a surrogate key. In many cases, it is desirable to include developer key attributes for foreign keys. For example, an employee service is used to process employees, and the employee has a department associated with it. The department has a DepartmentId (surrogate primary key) attribute and a DepartmentName (developer key) attribute.

When the consumer constructs the inbound payload, the consumer could choose to provide DepartmentId or DepartmentName or both. If only DepartmentName is provided, then logic needs to exist to resolve DepartmentId based on the name. If both name and id are provided, then validation must be provided to make sure the two matches. An LOV can be defined to resolve foreign key ID based on foreign alternate key attributes.
An LOV is defined at attribute level, but it can be configured to be driven by multiple attributes. When an LOV is driven by multiple attributes, the LOV query will use all the driven attributes in the where clause. Service framework doesn’t call individual setters, instead, it calls `setAttributeValues()` method in the `ViewRowImpl` class, which takes a list of attribute names and a second list of attribute values. LOVs will be fired after the attribute values are populated. So, if a foreign key has a composite alternate key, it will still work since the LOV fires after all the alternate key attributes are set.

When an attribute from a reference entity object is included in a view object, the attribute is read-only. However, if there is an LOV defined on the attribute, then the attribute is updateable in the sense that the attribute value can be used to drive the LOV.

The following are several scenarios and the steps you must perform for each of them.

**5.3.1.5.1 Scenario 1**

Single attribute Foreign Key and single attribute Alternate Key (Example: PersonVO has DeptId and Dname. DeptId is the foreign key ID, and Dname is the foreign alternate key).

An LOV should be defined on the alternate key attribute, with the foreign key ID as a derived attribute.

1. Create PersonVO based on PersonEO and a reference DeptEO. The foreign alternate key (Dname) from the reference EO is included in the PersonVO.
2. Define the LOV view object (DeptVVO).
3. Define a view accessor on PersonEO/PersonVO pointing to DeptVVO.
4. Define an LOV on the foreign alternate key (Dname) using the above view accessor, and configure the LOV to populate the foreign key ID (DeptId) as the derived attribute.

**5.3.1.5.2 Scenario 2**

Single attribute Foreign Key and multiple attribute Alternate Key (Example: PersonVO has OrganizationId as foreign key id, and OrganizationName+BusinessGroupName as the composite alternate key).

Each alternate key attribute needs to have an LOV defined, and each LOV should have all the alternate key attributes as the driving attribute and the foreign key ID as the derived attribute.

1. Create PersonVO based on PersonEO and a reference OrganizationEO and another reference BusinessGroupEO. The foreign alternate key (OrganizationName and BusinessGroupName) from the reference entity objects are included in the PersonVO.
2. Define the LOV view object (OrganizationVVO).
3. Define a view accessor on PersonEO/PersonVO pointing to OrganizationVVO.
4. Define an LOV on each of the foreign alternate key attributes (OrganizationName and BusinessGroupName) using the above view accessor, and configure the LOV to populate the foreign key ID (OrganizationId) as a derived attribute.
5. Modify `PersonVO.xml` to make the LOVs driven by all the foreign alternate key attributes. For example:
5.3.1.5.3 Scenario 3

Single attribute Foreign Key and multiple attribute Alternate Key and one of the alternate key attributes is another foreign key (Example: PersonVO has BirthOfCountry as foreign key id, and BirthOfCity as another foreign key ID. BirthOfCity has BirthOfCountry+CityName as a composite alternate key).

The first foreign key (BirthOfCountry) must be resolved first, either based on #1 or #2. Then the second alternate key should filter by the first alternate).

1. Create PersonVO based on PersonEO and a reference EO CountryEO and another reference EO CityEO. CountryName from CountryEO and CityName from CityEO should be included. CityName should be listed after CountryName in the PersonVO.

2. Define an LOV view object based on CountryEO.

3. Define an LOV view object based on CityEO. Define a view criteria to filter by CountryId.

4. Define a view accessor on PersonEO/PersonVO pointing to CountryVVO.

5. Define a view accessor on PersonEO/PersonVO pointing to CityVVO, and bind CountryId to BirthOfCountry.

6. Define an LOV on CountryName using CountryVVO view accessor, with BirthOfCountry as a derived attribute from CountryId from CountryVVO.

7. Define an LOV on CityName using CityVVO view accessor, with BirthOfCity as a derived attribute from CityId from CityVVO.

5.3.1.5.4 Scenario 4

Composite foreign key.

Each foreign key ID will be dealt with individually. For example, the foreign key id is OrgId+SourceId, then orgId and SourceId should be resolved based on solution in #1 or #2 or #3 separately. Then a validator must be defined to make sure combination of OrgId and SourceId is valid. This has the assumption that each individual attribute are a primary key itself.
5.3.2 Overview of Creating Services

The service interface is generated from an Oracle ADF application module. Go to the Service Interface tab in the Application Module wizard to expose the service interface for an application module.

5.3.2.1 What You May Need to Know About Design Time

This section discusses what happens during design time with application modules and the runtime object.

No Service Data Object in the Application Module

The custom methods in the Application Module do not take Data Object or a Data Object list as parameters. Instead, the Application Module's custom methods take ViewRowImpl/AttributeList or a list of ViewRowImpl/AttributeList as parameters. When you publish these methods in the service interface, ADF Business Components service will convert these to Data Object or a list of Data Object in the service interface during design time, and then performs conversion between Data Object and ViewRowImpl/AttributeList during runtime.

Return Object

The informational messages (and warnings) are reported as part of the return object. ADF Business Components generates appropriate wrappers as the return objects when necessary, and the wrappers contain the actual method return as well as the informational messages. Table 5-1 lists some examples:

<table>
<thead>
<tr>
<th>Table 5-1</th>
<th>Return Objects Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operation without Informational Messages (Support Warnings Flag is off)</strong></td>
<td><strong>Operation with Informational Messages (Support Warnings Flag is on)</strong></td>
</tr>
<tr>
<td>List&lt;Person&gt; processPerson(String op, List&lt;Person&gt; persons, ProcessControl ctrl)</td>
<td>PersonResult processPerson(String op, List&lt;Person&gt; persons, ProcessControl ctrl)</td>
</tr>
<tr>
<td>Person createPerson(Person person)</td>
<td>PersonResult createPerson(Person person)</td>
</tr>
<tr>
<td>void terminateEmployee(BigDecimal empId)</td>
<td>ServiceMessage terminateEmployee(BigDecimal empId)</td>
</tr>
<tr>
<td>String getApplicationName(BigDecimal applicationId)</td>
<td>StringResult getApplicationName(BigDecimal applicationId)</td>
</tr>
</tbody>
</table>

If the Support Warnings design time flag is off, no informational messages are returned (the first column in the above table). If the flag is on (the second column in the above table), then:
• `getXXX` returns the original object
• `create, update, mergeXXX, findXXX, and processXXX` returns the wrapper object that contains a list of the original object and a list of information messages
• `deleteXXX` returns the informational message
• Each custom method can be configured individually about whether to return informational messages

Note:
Note that `ServiceMessage` is created by the framework based on the warning or errors thrown during service execution. When you develop your service, you don’t throw `ServiceMessage`. Instead, you throw `JboException` or `ApplcoreException`. See About Defining Service Exceptions and Information for information about error and informational messages.

Find Operation
When you include the standard operations in a service interface, you can enable the generic find operation, or select a view criteria and expose a find operation that uses the view criteria.

Oracle ADF has a default list of operators (such as =, contains, and so on) that can be used in the find operation. You can enable more custom operators or disable an operator in the default list.

5.3.3 How to Generate Synchronous and Asynchronous Service Methods

Each service method can be exposed as both synchronous version and asynchronous version.

To generate synchronous and asynchronous services methods, do the following:

1. Go to the Application Module Design Time wizard.
2. Choose the Service Interface tab.
3. Choose the Service Interface category. See Figure 5-2.

Figure 5-2 Edit Service Interface Dialog

4. Select Generate Asynchronous Web Service Methods. Click OK.
5. Save your changes.
5.3.4 About Exposing Flexfields

Flexfields are used as extension points. If your object has a flexfield defined, then you might want to consider including the flexfield in the service interface so that your customer can utilize flexfields through web service.

See the following sections for information about how to expose the flexfields in a service interface:

- Publishing Descriptive Flexfields as Web Services
- Making Extensible Flexfields Available to Web Services
- How to Publish Key Flexfield Application Modules as Web Services

5.3.5 About Enabling Security

You need to enable security on your service to make sure only the granted people can invoke the service.

5.3.5.1 Authentication

An authentication policy determines how the caller proves its identity. A simple case could be user name and a password in clear text, which is simple but not very secure. Or a token can be generated by an identity provider and then passed to the service provider.

See Securing Web Services Use Cases for information about the different authentication policies supported. Once a security policy is chosen, you need to add it as an annotation in your xxxServiceImpl class. For example:

```java
@SecurityPolicy( { "oracle/wss11_saml_or_username_token_with_message_protection_service_policy"})
@CallbackSecurityPolicy("oracle/wss11_saml_token_with_message_protection_client_policy")
```

5.3.5.2 Enable Authorization

Authorization determines who can invoke a service operation. To enable authorization, do the following:

1. Add ServicePermissionCheckInterceptor as one of the EJB interceptors in your xxxServiceImpl class. The result will be similar to this:

```java
@Interceptors({ServiceContextInterceptor.class,
               ServicePermissionCheckInterceptor.class})
```

**Note:**

ADF Business Components service is implemented with EJB, and the EJB interceptor is used here to authorize before any service operation can be invoked.

2. Grant access to the service operations to desired roles in jazn-data.xml:
   a. Add a web service resource type.
Navigate to Application Resources >Descriptors/META-INF/jazn-data.xml, click the Resource Grants tab, and then add a new resource type. This is shown in Figure 5-3.

**Figure 5-3  Create Resource Type**

![Create Resource Type](image)

The following xml snippet in **jazn-data.xml** will be generated:

```xml
<resource-type>
    <name>WebserviceResourceType</name>
    <display-name>WebserviceResourceType</display-name>
    <description>Webservice Resource</description>
    <matcher-class>oracle.wsm.security.WSFunctionPermission</matcher-class>
    <actions-delimiter>,'</actions-delimiter>
    <actions>invoke</actions>
</resource-type>
```

b. Add a resource, as shown in Figure 5-4.

- Add the privilege and specify which role can access the resource defined above.
- Click Entitlement Grants in **jazn-data.xml**.
- Create a new entitlement or edit an existing one.
- Add the newly created resource.
- Choose the action.
c. Add JpsInterceptor in ejb-jar.xml.

Note:

In the previous steps, the authorization is granted to the application roles. The application roles are computed using JpsInterceptor. Without the JpsInterceptor setting in ejb-jar.xml, the application roles won’t be computed and authorization will fail.

- Navigate to service project > Application Sources > META-INF.ejb-jar.xml and add the following code snippet:

```xml
<ejb-jar...>
  ...
  <interceptors>
    <interceptor>
      <interceptor-class>oracle.security.jps.ee.ejb.JpsInterceptor</interceptor-class>
      <env-entry>
        <env-entry-name>application.name</env-entry-name>
        <env-entry-type>java.lang.String</env-entry-type>
        <env-entry-value><application_name></env-entry-value>
        <injection-target>
          <injection-target-class>oracle.security.jps.ee.ejb.JpsInterceptor</injection-target-class>
          <injection-target-name>application_name</injection-target-name>
        </injection-target>
      </env-entry>
    </interceptor>
  </interceptors>
  ...
</ejb-jar>
Replace *application_name* with your application name, which is defined under `policy-store/applications/application/name` element in your `jazn-data.xml`.

d. Specify the applicationid in `weblogic-application.xml`:

```xml
<application-param>
  <param-name>jps.policystore.applicationid</param-name>
  <param-value><application_name></param-value>
</application-param>
```

Replace *application_name* with the application name you used in Step 2.c.

### 5.3.6 About Using the Java Transaction API

Java Transaction API (JTA) is used to handle ADF Business Components transactions.

#### 5.3.6.1 Specify the Data Source

JTA manages distributed transactions across multiple resources. However, the resources must participate in global transaction to be part of the JTA. All the ADF Business Components services must use ApplicationServiceDBDS as the data source, and must not use ApplicationDBDS. ApplicationDBDS doesn't participate global transaction. ApplicationServiceDBDS supports global transaction using Oracle WebLogic Server Emulate Two-Phase Commit emulation.

To specify the data source used by the service, do the following:

1. Click the **Configuration** tab of Application Module wizard, then choose the configuration with type "SI", as shown in Figure 5-5.
2. Select the appropriate data source for your service.

If you don’t see the data source in the dropdown list, you will need to create the database connection first.

3. Navigate to `ejb-jar.xml` and add ApplicationServiceDBDS as the `resource-ref`:

   ```xml
   <ejb-jar ...>
   <enterprise-beans>
   <session>
   <ejb-name>.....</ejb-name>
   <resource-ref>
   <res-ref-name>jdbc/ApplicationDBDS</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <res-auth>Container</res-auth>
   </resource-ref>
   <resource-ref>
   <res-ref-name>jdbc/ApplicationServiceDBDS</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <res-auth>Container</res-auth>
   </resource-ref>
   ....
   ```

Related Links
The following document provides additional information related to subjects discussed in this section:

- For more information about data sources, see "JDBC Data Source Transaction Options" in Administering JDBC Data Sources for Oracle WebLogic Server.

### 5.3.6.2 About Transaction Attributes

ADF Business Components service is implemented as a stateless session bean. Transaction attributes can be specified at the class level or method level, and these attributes determine the JTA behavior.

### 5.3.7 About Deploying Services

The service can be deployed to an integrated or standalone Oracle WebLogic Server service.

To deploy to an integrated Oracle WebLogic Server, right-click the xxxServiceImpl class, and then select **Run** from the context menu.

To deploy to the standalone Oracle WebLogic Server, create a Business Component Service Interface profile. The window is shown in

**Figure 5-6  Create Deployment Profile**

![Create Deployment Profile](image)

This profile is a compound profile and includes two child profiles: Common (JAR File) and MiddleTier (EJB Jar File). The Common profile includes the service interface. This .jar file will be required by the consumer when the consumer uses ServiceFactory to invoke the service. (See Using Service Factory for more information.) The MiddleTier .jar file contains the service implementation. MiddleTier profile has a dependency on the common profile.

Include the MiddleTier profile in your application’s ear profile (Ear Deployment Profile >Application Assembly) so that the service will be included in the application ear file. Do not include the Common profile in the ear profile’s Application Assembly. The Common profile will be included via profile dependency.

**Note:**

Do not include connections.xml in the common .jar file. There cannot be more than one connections.xml in one application, otherwise whatever shows up first in the classpath will be picked up, and this one might not be the right one. To prevent that, the service interface common profile cannot include connections.xml.
5.3.7.1 About Service Context Root

When the service is deployed, the service can be accessed through a service end point URL, such as http://localhost:7101/mycontext/MyService. Note that the context root of a service is defined at the service project (Project properties >Java EE Application > Java EE Web Context Root, shown in Figure 5-7).

Figure 5-7 Project Properties

5.3.8 About Testing Services

You can test the service only after it has been deployed.

5.3.8.1 Understanding What to Test

An ADF Business Components service is based on ADF Business Components. These components should have been already tested with junit, such as the defaulting and validation logic, queries, and other business logic. One commonly asked question is: Why should I test my service since my underlying application module/view object has been tested already? The ADF Business Components service testing is not a duplicate test of the ADF Business Components testing. Instead, the focus should be on the service layer and what is not covered in the ADF Business Components testing.

Areas that are not covered in the ADF Business Components testing are the following:

- Security: ADF Business Components service is authenticated and authorized at the service layer, not the business components object layer.

- Exception: The exception will come back as soap fault instead of JboException. You should check it has all the details that you expected.

However, you do not need to repeat the tests that have been done in ADF Business Components testing:
• Data security is defined at the business components object layer, so you do not need to repeat the data security testing at service layer.

• You need to test each service method, but you do not need to test all permutations of inbound parameters. Therefore, you probably just need to test a success case and a couple of typical failure cases for each of your service methods.

The following is a list of items that you should validate during the service testing (both the synchronous and asynchronous versions).

• If the service is running
• If the service is responding within a reasonable amount time
• For all service methods:
  – Successful cases: Verify the response. You might also want to do a query after a post to check the data is indeed committed.
  – Failure cases: Verify the fault comes back, and with the expected content.
  – Security: Test both authentication and authorization.

5.3.8.2 Understanding How to Test

For ad-hoc testing, use the browser test page, HTTP Analyzer, or Oracle Enterprise Manager.

• Web service test page

In a web browser, access the service endpoint URL. The service end point URL follows the format of http://host:port/context_root/service_name. When the service is deployed to integrated Oracle WebLogic Server, the service endpoint URL is printed out in the Oracle JDeveloper console.

After you open up the web service test page, you can enter the inbound parameters from the UI and submit the service request. For the service that is secured with GPA, you will need to expand the WS-Security element, and provide the user name and password there. For service that is secured with oracle/wss11_saml_or_username_token_with_message_protection_service_policy, you cannot invoke it from the web service test page.

• HTTP Analyzer

You can access HTTP Analyzer within Oracle JDeveloper by navigating to Tools > HTTP Analyzer, and then clicking on "Create New Request". You then can follow the UI to invoke a service. Similarly, it doesn’t really handle service secured with oracle/wss11_saml_or_username_token_with_message_protection_service_policy.

• Oracle Enterprise Manager

Use Oracle Enterprise Manager to invoke a service. For more information, see the Oracle Fusion Middleware Administering Web Services guide.

For unit testing, you should write unit test code.

http://globaldc.oracle.com/perl/twiki/view/FusionSharedTools/ApplicationsTestFramework
5.4 About Invoking Services

All ADF Business Components services have both synchronous and asynchronous versions for the same method. The service consumer must decide which version of the service method to use.

A service method can be invoked synchronously if all of the following conditions are met:

- The invoked method takes a simple payload, such as a single document or a fixed number of documents (including parent and children).
- The invoked method is expected to be finished in real time and takes no longer than a few seconds.

The consumer should consider invoking the method asynchronously if one of the following conditions is met:

- The method takes a flexible number of documents, such as a list of service data objects.
- The method may be long-running.

5.4.1 Invoking a Synchronous Service

You can invoke a synchronous service using service factory, service-based entity object and view object, Java API for XML Web Services (JAX-WS) client, or from SOA.

5.4.1.1 Using Service Factory

If you need to invoke a synchronous service from a Java client, including an ADF Business Components component, UI, or Oracle Enterprise Scheduler, then using a service factory is recommended.

- It is easier to write a service client using service factory than using JAX-WS.
- The consumer side doesn’t need to generate or maintain the source control of the proxy code.
- If the service is co-located, the service invocation is more performant because it does not invoke XML serialization and de-serialization.

To invoke a service using service factory, do the following:

1. Identify the common.jar file provided by the service provider team.
   Service factory requires that the service interface common.jar file be in the classpath of the consumer. (See About Deploying Services for information service deployment.) Subsequently, the consumer needs to identify the name of the common.jar file first from the provider team, and include it in the classpath.

2. Retrieve the service endpoint information from the connections.xml file.
   Service Factory can be used to invoke a service via Remote Method Invocation (RMI) or Simple Object Access Protocol (SOAP). RMI can only be used within one domain, for example, receivables application invokes a service deployed in ledger application. RMI cannot be used across an Oracle WebLogic Server domain due to security constraint. For the cross Oracle WebLogic Server domain service
invocation, such as when receivables invokes a service hosted in Oracle Fusion Human Capital Management, SOAP must be used.

Service Factory uses ADF Connection Architecture to retrieve the service endpoint information from connections.xml. It creates a dynamic proxy to the remote service using the designated protocol in connections.xml.

The following is a sample entry in connections.xml using RMI:

```xml
<Reference name="{http://xmlns.oracle.com/apps/sample/dtsvc/}WorkerService" className="oracle.jbo.client.svc.Service" xmlns="">
    <Factory className="oracle.jbo.client.svc.ServiceFactory"/>
    <RefAddresses>
        <StringRefAddr addrType="serviceInterfaceName">
            <Contents>oracle.apps.sample.dtsvc.WorkerService</Contents>
        </StringRefAddr>
        <StringRefAddr addrType="serviceEndpointProvider">
            <Contents>ADFBC</Contents>
        </StringRefAddr>
        <StringRefAddr addrType="jndiName">
            <Contents>WorkerServiceBean#oracle.apps.sample.dtsvc.WorkerService</Contents>
        </StringRefAddr>
        <StringRefAddr addrType="serviceSchemaName">
            <Contents>WorkerService.xsd</Contents>
        </StringRefAddr>
        <StringRefAddr addrType="serviceSchemaLocation">
            <Contents>oracle/apps/sample/dtsvc/</Contents>
        </StringRefAddr>
        <StringRefAddr addrType="jndiFactoryInitial">
            <Contents>weblogic.jndi.WLInitialContextFactory</Contents>
        </StringRefAddr>
        <StringRefAddr addrType="jndiProviderURL">
            <Contents>t3://localhost:7202</Contents>
        </StringRefAddr>
    </RefAddresses>
</Reference>
```

In the example, the serviceEndpointProvider is ADF Business Components, which indicates that it is a RMI invocation.

You need to replace the code in bold with the service that you plan to invoke. Table 5-2 lists the appropriate values:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference/name</td>
<td>The service qualified name. It follows the format of {&lt;targetNamespace&gt;}&lt;name&gt;. The targetNamespace and name can be found in the wsdl file of the service to be invoked.</td>
<td>({<a href="http://xmlns.oracle.com/apps/sample/dtsvc/%7DWorkerService">http://xmlns.oracle.com/apps/sample/dtsvc/}WorkerService</a>)</td>
</tr>
</tbody>
</table>
### Table 5-2 (Cont.) Service Property Values

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>serviceInterfaceName</td>
<td>This is the service interface name. You can get it from the service interface java class in the common .jar file.</td>
<td>oracle.apps.sample.dtsvc.WorkerService</td>
</tr>
<tr>
<td>jndiName</td>
<td>This is the EJB bean name. ADF Business Components service is implemented by EJB. It follows the format of &lt;ServiceName&gt;Bean#&lt;ServiceInterfaceName&gt;</td>
<td>WorkerServiceBean#oracle.apps.sample.dtsvc.WorkerService</td>
</tr>
<tr>
<td>serviceSchemaName</td>
<td>The ADF Business Components service always has a schema file generated for the service. It follows the format of &lt;ServiceName&gt;.xsd.</td>
<td>WorkerService.xsd</td>
</tr>
<tr>
<td>serviceSchemaLocation</td>
<td>The path of the service schema. It is basically the service interface package path.</td>
<td>oracle/apps/sample/dtsvc/</td>
</tr>
<tr>
<td>jndiProviderUrl</td>
<td>The host name and port number where the service is deployed. This is not needed during design time, but apparently required during runtime. This value will be tokenized during packaging and be &quot;replaced&quot; with the server URL during provisioning.</td>
<td>t3://localhost:7202</td>
</tr>
</tbody>
</table>

For a SOAP invocation, two entries are required in connections.xml: a ServiceFactory entry and a web service connection entry. The first entry is very similar to the RMI entry except for the following:

- The serviceEndPointProvider value is SOAP.
- A webServiceConnectionName entry, which is used to link to the second entry, is included. That is, the value of webServiceConnectionName must be the name of the second connection.
- There are no jndiFactoryInitial or jndiProviderUrl properties.
Example 5-2 and Example 5-3 show the two entries in connections.xml using SOAP.

The second connection is a standard web service connection. To generate the web service connection entry in connections.xml, do the following:

a. Create a web service proxy. (See "Creating a Web Service Proxy Class to Programatically Access the Service" in Developing Fusion Web Applications with Oracle Application Development Framework, for more information.)

b. Create a web service connection. (See "How to Create a New Web Service Connection" in Developing Fusion Web Applications with Oracle Application Development Framework for more information.)

c. Remove the web service proxy code.

---

**Note:**

The connections.xml file in source control contains the concrete web service endpoint URL. This URL can be used for testing in integrated Oracle WebLogic Server. As part of the ear generation, these URLs will be looked up and replaced with abstract tokens. These tokens will again be replaced when application is installed at customer site with the customer's server URL.

There is a ServiceRepository.xml file that contains the list of services produced by Oracle Fusion Applications. The token replacement during ear generation uses this file to look up and replace the concrete URL with abstract tokens.

---

3. Secure the service.

For RMI invocation, only identity propagation is supported. Subsequently, there is nothing more that you need or can configure.

For SOAP invocation, remember that the client-side security policy is driven by the server-side security policy, and the client-side policy must match the server-side policy. In most cases, you can utilize GPA, which uses wss10_saml_token_client_policy and propagates the identity from the caller to the service. The following cases require the use of LPA:

- The service is protected by message protection policy. Usually, this service is an external service that can be accessed outside the Fusion Applications firewall. In this case, you will need to add the security policy in the web service connection of the connections.xml. For example:

```xml
<Reference name="HrServiceConnection" 
className="oracle.adf.model.connection.webservice.impl.WebServiceConnectionImpl" xmlns="">
  <Factory 
className="oracle.adf.model.connection.webservice.api.WebServiceConnectionFactory"/>
  <RefAddresses>
    <XmlRefAddr addrType="WebServiceConnection"> 
      <Contents> 
        <wsconnection ......> 
          <model ......>
            <service ......>
              <port ......> 
                <policy-references xmlns="http://oracle.com/af"/>
```
The following are the client-side policies that you can use to invoke a service protected by message protection policy:

- wss_username_token_over_ssl_client_policy
- wss_saml_token_bearer_over_ssl_client_policy
- wss11_username_token_with_message_protection_client_policy
- wss11_saml_token_with_message_protection_client_policy
- wss_http_token_over_ssl_client_policy

- Identity switch is required. If the service must be invoked with a different user (usually using appid), then you need to specify LPA:

```xml
<call-properties xmlns="http://oracle.com/adf">
  <call-property id="csf-key" xmlns=""/>
</call-properties>

<soap ...

Replace system1-key with the csf-key holding the application identity user name and password required by the user case. The actual application identities and their corresponding csf-keys are centrally provided.

If the service is protected by message protection policy, then use oracle/wss11_username_token_with_message_protection_client_policy.

4. Invoke the service.

Write your client code to invoke the service. For example,

```java
OrganizationService svc = (OrganizationService) ServiceFactory.getServiceProxy(OrganizationService.NAME);
DataFactory datafactory = ServiceFactory.getDataFactory(svc);
List orgs = new ArrayList(2);
Org org1 = (Org) datafactory.create(Org.class);
org1.setOrgName("OrgName");
org1.setName("TranslatedName");
org1.setDescription("Your org Description"); //... and set more attributes
orgs.add(org1);
svc.processOrganization("Merge", orgs, null);
```
SDOs based on polymorphic view objects don’t have Java classes generated. To create these SDOs, use the generic DataObject class:

```java
import commonj.sdo.DataObject;

DataObject emp = datafactory.create("http://xmlns.oracle.com/apps/fooService/", "Bar");
emp.set("Empno", new Long(8080));
emp.set("Ename", "Oliver");
emp.set("Job", "MANAGER");
```

**Note:**
You will need to add "BC4J Service Client" library to your client project.

5. Perform runtime control for the query.

The standard `find` method API provides control over the query behavior.

- **Partial Attribute**
  By default, the find operation returns all the attributes including all details. When you only need some attributes, you should set the partial attributes on the `FindCriteria` parameter of the `find` method. Do this in the following situations:
  - SDO contains LOB, which can be very expensive to retrieve and transfer.
  - SDO contains details that are not needed, such as translations. Querying detail is also expensive.

**Note:**
The standard `getXXX` function doesn’t take a `FindCriteria`, so this function always returns everything. You should use `findXXX` to trim your return attributes.

The following example shows how to set the partial attributes to include only `Dname`, `Loc` from `Dept`, and exclude `Empno` from `Emp`:

```java
FindCriteria fc = (FindCriteria)datafactory.create(FindCriteria.class);
List l = new ArrayList();
l.add("Dname");
l.add("Loc");
l.add("Emp");
fc.setFindAttribute(l);
List cfcl = new ArrayList();
ChildFindCriteria cfc =
    (ChildFindCriteria)datafactory.create(ChildFindCriteria.class);
cfc.setChildAttrName("Emp");
List cl = new ArrayList();
cl.add("Empno");
cfc.setFindAttribute(cl);
cfc.setExcludeAttribute(true);
cfcl.add(cfc);
f.setChildFindCriteria(cfcl);
DeptResult res = svc.findDept(fc, null);
```
The following example shows how to set the partial attributes to exclude PurchaseOrderLine from PurchaseOrder:

```java
FindCriteria fc = (FindCriteria)datafactory.create(FindCriteria.class);
List l = new ArrayList();
fc.setExcludeAttribute(true);
l.add("PurchaseOrderLine");
fcc.setFindAttribute(l);
PurchaseOrderResult res = svc.findPurchaseOrder(fc, null);
```

**Filter**

The find API allows you to specify the WHERE clause of your query. The WHERE clause can be set on any level of the SDO. The following example shows how to retrieve only the departments with a department number greater than 10 and child employees whose names start with 'A':

```java
FindCriteria fc = (FindCriteria)datafactory.create(FindCriteria.class);
//create the view criteria item
List value = new ArrayList();
value.add(new Integer(10));
ViewCriteriaItem vci = (ViewCriteriaItem)datafactory.create(ViewCriteriaItem.class);
vci.setValue(value);
vci.setAttribute("Deptno");
List<ViewCriteriaItem> items = new ArrayList(1);
items.add(vci);
//create view criteria row
ViewCriteriaRow vcr = (ViewCriteriaRow)
datafactory.create(ViewCriteriaRow.class);
vcr.setItem(items);
//create the view criteria
List group = new ArrayList();
group.add(vcr);
ViewCriteria vc = (ViewCriteria)datafactory.create(ViewCriteria.class);
vc.setGroup(group);
//set filter
fc.setFilter(vc);
```

List cfc1 = new ArrayList();
//create the child find criteria
ChildFindCriteria cfc = (ChildFindCriteria)datafactory.create(ChildFindCriteria.class);
cfc.setChildAttrName("Emp");
//create the child view criteira
ViewCriteria cvc = (ViewCriteria)datafactory.create(ViewCriteria.class);
cfc.setFilter(cvc);
//create the view criteria item
List cvalue = new ArrayList();
cvalue.add("A%");
ViewCriteriaItem cvci = (ViewCriteriaItem)datafactory.create(ViewCriteriaItem.class);
cvci.setValue(value);
cvci.setAttribute("Dname");
cvci.setOperator("LIKE");
List<ViewCriteriaItem> citems = new ArrayList(1);
citems.add(cvci);
//create child view criteria row
ViewCriteriaRow cvcr = (ViewCriteriaRow)
datafactory.create(ViewCriteriaRow.class);
cvcr.setItem(citems);
List cgroup = new ArrayList();
cgroup.add(cvcr);
cvc.setGroup(cgroup);

DeptResult dres = svc.findDept(fc, null);

You also can query the parents with the children that satisfy certain criteria. For example, use the following to retrieve the departments with employees whose salary is greater than $10,000:

//create the view criteria item on the employees
List nvalue = new ArrayList();
nvalue.add(new BigDecimal(10000));
ViewCriteriaItem nvci = (ViewCriteriaItem)datafactory.create(ViewCriteriaItem.class);
vci.setValue(nvalue);
vci.setAttribute("Salary");
vci.setOperation(">");
List<ViewCriteriaItem> nitems = new ArrayList(1);
nitems.add(nvci);
//create view criteria row
ViewCriteriaRow nvcr = (ViewCriteriaRow)
datafactory.create(ViewCriteriaRow.class);
nvcr.setItem(nitems);
//create the nested view criteria
List ngroup = new ArrayList();
group.add(nvcr);
ViewCriteria nvc = (ViewCriteria)datafactory.create(ViewCriteria.class);
nvc.setGroup(ngroup);

//create the view criteria item on the department
ViewCriteriaItem vci = (ViewCriteriaItem)datafactory.create(ViewCriteriaItem.class);
vci.setAttribute("Emp");
vci.setNested(nvc);
List<ViewCriteriaItem> items = new ArrayList(1);
items.add(vci);
//create view criteria row
ViewCriteriaRow vcr = (ViewCriteriaRow)
datafactory.create(ViewCriteriaRow.class);
vcr.setItem(items);
//create view criteria on department
ViewCriteria vc = (ViewCriteria)datafactory.create(ViewCriteria.class);
group = new ArrayList()
group.add(vcr);
vcr.setGroup(group);
//set filter
FindCriteria fc = (FindCriteria)datafactory.create(FindCriteria.class);
fcs.setFilter(vc);

DeptResult dres = svc.findDept(fc, null);

• Paging

If you know that your query might return a large amount of data, you should make multiple service invocations, and use FetchSize and FetchIndex to control the amount of the data that you want to retrieve.

The following example shows how to retrieve only the second employee and the employee's department:

FindCriteria fc = (FindCriteria)datafactory.create(FindCriteria.class);
List cfcl = new ArrayList();
ChildFindCriteria cfc =
(ChildFindCriteria)datafactory.create(ChildFindCriteria.class);
cfc.setChildAttrName("Emp");
cfc.setFetchStart(1);
cfc.setFetchSize(1);
cfcl.add(cfc);
fc.setChildFindCriteria(cfcl);
DeptResult dres = svc.findDept(fc, null);

6. Perform runtime control for post.

The process API provides a bulk operation that can handle multiple SDOs simultaneously.

- Return mode
  Use ReturnMode on ProcessControl to specify whether you want to return a list of SDOs with all attributes, or with primary key attributes, or return nothing in processXXX method. You should only return the primary key or nothing, unless you need the full SDO returned for further processing.

- Exception return mode
  When an error occurs, the returned error message can contain just the primary key of the failure SDO, or all attributes of the failure SDO. You can specify which behavior you want using the ExceptionReturnMode attribute of the ProcessControl parameter in the processXXX method.

- Partial failure
  For bulk load, it often makes more sense to commit as many records as possible, and report the problematic data. By default, either all SDOs go through or none goes through in processXXX method. However, you can call setPartialFailureAllowed on ProcessControl and pass that ProcessControl to the processXXX method. This will turn on the partial failure feature.

7. Handle exceptions.

All ADF Business Components services throw ServiceException. ServiceException contains FaultInfo, and FaultInfo stores the error messages. (Based on JAX-WS 2.0, exceptions thrown from service operations need to have a property named "FaultInfo").

@WebFault(name="ServiceErrorMessage", targetNamespace="http://xmlns.oracle.com/adf/svc/errors/",
          faultBean="#oracle.jbo.service.errors.ServiceErrorMessage")
public class ServiceException extends RuntimeException
{
    public ServiceErrorMessage getFaultInfo()...
    public void setFaultInfo(ServiceErrorMessage faultInfo)...
}

The ServiceErrorMessage can contain one or more child error messages, and each child can contain its own children. Basically it is a tree hierarchy. You could consider ServiceErrorMessage as a counterpart of the JboException at the service layer.
The ServiceErrorMessage has a few subclasses including ServiceAttrValErrorMessage, ServiceRowValErrorMessage, and ServiceDMLErrorMessage. Their counterparts in ADF Business Components are AttrValException, RowValException, and DMLException. Note that ServiceException is an Exception, but all other classes above including ServiceMessage, ServiceErrorMessage, ServiceAttrValErrorMessage, ServiceRowValErrorMessage, and ServiceDMLErrorMessage are not exceptions.

As the consumer of a service, if you call a service and need to take different actions based on the exceptions, you might need to walk through the message tree structure.

The following is a simple example that walks through the message and converts to JboException:

```java
public static RuntimeException convertServiceException(ServiceException ex) {
    return convertToRuntimeException(ex.getFaultInfo());
}

public static RuntimeException convertToRuntimeException(ServiceMessage svcMsg) {
    RuntimeException ret = null;
    String msgStr = svcMsg.getMessage();
    List d = svcMsg.getDetail();
    Throwable[] details = null;
    if(d != null && !d.isEmpty()) {
        details = new Throwable[d.size()];
        for(int i=0; i<d.size(); i++) {
            Object obj = d.get(i);
            if(obj instanceof ServiceMessage)
                details[i] = convertToRuntimeException((ServiceMessage)obj));
            else if(obj instanceof Throwable)
                details[i] = (Throwable) obj;
            else
                details[i] = new Throwable(obj.toString());
        }
    }
    if(svcMsg instanceof ServiceErrorMessage) {
        JboException ex = new JboException(msgStr);
        ex.setExceptions(details);
        ret = ex;
    }
    else {
        JboWarning w = new JboWarning(msgStr);
        w.setDetails(details);
        ret = w;
    }
    return ret;
}
```

8. Create informational messages.

Informational messages are not exceptions, and won’t stop service from committing the current transaction. However, these information might be very useful to the clients. For example, you might want to know when there is not enough money in your checking account and have the system automatically transfer some from your savings account.
The service provides decides whether a service will possibly return informational messages or not. If so, the informational message will be part of the return, as shown below:

```java
public EmployeeResult processEmployee(String changeOperation,
    List<Employee> employee,
    ProcessControl processControl) throws
ServiceException;
```

As the service consumer, you should retrieve the informational message from the return and perform appropriate actions such as converting it to `JboException` and displaying it in the UI. An example might be:

```java
ProcessControl pc = (ProcessControl)datafactory.create(ProcessControl.class);
pc.setPartialFailureAllowed(true);
EmployeeResult res = svc.processEmployee("Create", list, pc);
if(res != null) {
    List msgs = res.getMessage();
    if(svcMsgs != null && !svcMsgs.isEmpty()) {
        Exception[] exceptions = new Exception[svcMsgs.size()];
        for(int i=0; i<exceptions.length; i++)
            exceptions[i] = convertToRuntimeException(svcMsgs.get(i));
        JboException ex = new JboException(....., exceptions); //create a bundled
            exception with the service errors as detail
        }
    throw ex;
}
```

For more information about invoking a service using service factory, see
Synchronously Invoking an ADF Business Components Service from an Application.

Example 5-2 Entry 1

```xml
<Reference name="[http://xmlns.oracle.com/apps/sample/hrService/]HrService" 
className="oracle.jbo.client.svc.Service" xmlns="">
    <Factory className="oracle.jbo.client.svc.ServiceFactory"/>
    <RefAddresses>
        <StringRefAddr addrType="serviceInterfaceName">
            <Contents>oracle.apps.sample.hrService.HrService</Contents>
        </StringRefAddr>
        <StringRefAddr addrType="serviceEndpointProvider">
            <Contents>SOAP</Contents>
        </StringRefAddr>
        <StringRefAddr addrType="webServiceConnectionName">
            <Contents>HrServiceConnection</Contents>
        </StringRefAddr>
        <StringRefAddr addrType="serviceSchemaName">
            <Contents>HrService.xsd</Contents>
        </StringRefAddr>
        <StringRefAddr addrType="serviceSchemaLocation">
            <Contents>oracle/apps/sample/hrService/</Contents>
        </StringRefAddr>
    </RefAddresses>
</Reference>
```
Example 5-3 Entry 2

<Reference name="HrServiceConnection"
className="oracle.adf.model.connection.webservice.impl.WebServiceConnectionImpl"
xmlns=""">
  <Factory
className="oracle.adf.model.connection.webservice.api.WebServiceConnectionFactory"/>
  <RefAddresses>
    <XmlRefAddr addrType="WebServiceConnection">
      <Contents>
        <wsconnection description="http://rws65094fwks:7202/MySampleSvc/HrService?WSDL" service="{http://xmlns.oracle.com/apps/sample/hrService/}HrService">
          <model name="{http://xmlns.oracle.com/apps/sample/hrService/}HrService" xmlns="http://oracle.com/ws/model">
            <service name="{http://xmlns.oracle.com/apps/sample/hrService/}HrService">
              <port name="HrServiceSoapHttpPort" binding="{http://xmlns.oracle.com/apps/sample/hrService/}HrServiceSoapHttp" portType="{http://xmlns.oracle.com/apps/sample/hrService/}HrService">
                <policy-references xmlns="http://oracle.com/adf">
                  <policy-reference category="security" uri="oracle/wss11_saml_token_with_message_protection_client_policy" enabled="true" id="oracle/wss11_saml_token_with_message_protection_client_policy" xmlns=""/>
                </policy-references>
                <soap addressUrl="http://rws65094fwks:7202/MySampleSvc/HrService" xmlns="http://schemas.xmlsoap.org/wsdl/soap/"/>
              </port>
            </service>
          </model>
        </wsconnection>
      </Contents>
    </XmlRefAddr>
  </RefAddresses>
</Reference>

5.4.1.2 About Using Service-Based Entity Object and View Object

When you need to work with output from a service in the format of an ADF Business Components component, such as rendering the data in a UI table or creating a view link to it, then you should consider using service-based entity objects and view objects.

For more information about working with data from a remote ADF Business Components service, see Working with Data from a Remote ADF Business Components Service.

5.4.1.3 About Using the JAX-WS Client

Generally, you should not use JAX-WS or Java APIs for XML-Based Remote Procedure Call (JAX-RPC) client to access an ADF Business Components service. You can use JAX-WS to access BPEL or a third-party service.

See "Creating a Web Service Proxy Class to Programmatically Access the Service" in Developing Fusion Web Applications with Oracle Application Development Framework for information about how to create a JAX-WS proxy.

In summary, you need to do the following

- Create the JAX-WS proxy client.
  
  Navigate to New Gallery > Business Tier > Web Services > Web Service Proxy to create proxy clients.
Note:

Perform the steps as listed when using the web proxy wizard to create the JAX-WS proxy client, with the following exceptions:

- At the "Select Web Service Description" screen, deselect the **Copy WSDL into Project** checkbox.
- At the "Specify Default Mapping Options" screen, deselect the **Generate As Async** checkbox.

For the proxy code, it is recommended to have proxy in the package structure, such as `oracle.apps.<lbaTop>.<lbaCore>.<xyzService>.proxy`. It is recommended to generate the type in the "type" subpackage.

- Create a web service connection.

  The proxy generated in previous step hard coded the service end point. To externalize the end point to `connections.xml`, you need to right-click on the generated proxy object, and choose **Create Web Service Connection**. In the Edit Web Service Connection dialog, provide the appropriate information, but don’t provide any value for user name or password.

  This will create a connection entry in `connections.xml`. This entry can be changed during deployment.

  You can attach different security policies in the `connections.xml` file, as described in the Using Service Factory.

- Develop client-side code. For example:

  ```java
  ADFContext aDFContext = ADFContext.getCurrent();
  WebServiceConnection con = (WebServiceConnection)aDFContext.getConnectionsContext().lookup("YourConnectionNameHere");
  YourService svc = con.getJaxWSPort(YourService.class);
  //invoke your service below
  ```

5.4.1.4 About Using SOA

When you invoke an ADF Business Components service from BPEL, you usually use the asynchronous version unless you are sure the service satisfies the synchronous invocation condition that was discussed previously.

For more information, see Common Service Use Cases and Design Patterns.
Caution:
Web Service data control is an anti-pattern that is not allowed. Instead, build ADF Business Components components and bind these components to the UI. If you are trying to invoke an ADF Business Components service, then you can use service-enabled entity objects and view objects.

For more information, see the "How to Service-Enable Individual View Objects" section in the "Integrating Service-Enabled Application Modules" chapter of the *Developing Fusion Web Applications with Oracle Application Development Framework*.

PL/SQL calling Web Service is also an anti-pattern that is not allowed because of security issues.

5.4.2 Understanding How to Invoke an Asynchronous Service

In general, BPEL is used to invoke an asynchronous service. If you need to invoke an ADF Business Components service from Java that does not meet the synchronous condition, then you must use one of the following alternate approaches:

**Asynchronous Invocation (The caller-side must wait for response)**

- **If the caller is ADF UI**: The UI must raise an event, which is received by a mediator. The mediator invokes a BPEL process that invokes the asynchronous service and then invokes a second service after receiving the callback. The second service is responsible for notifying the UI side that the process has completed and then the UI uses the Active Data Service to refresh the UI.

  For information about how to enable the UI for dynamic update via Active Data Service, see Implementing an Asynchronous Service Initiation with Dynamic UI Update.

- **If the caller is Oracle Enterprise Scheduler Service**: Oracle Enterprise Scheduler Service Java Jobs can invoke the asynchronous service via a JAX-WS proxy, but must set the asynchronous callback service to that of the Oracle Enterprise Scheduler Service Web Service. During this time, the Job’s status will be Running and when the asynchronous callback comes through the Oracle Enterprise Scheduler Service Web Service callback port, the Job code will be notified with the response and can Complete.

**One-way Invocation (The caller fires and forgets)**

- The caller must raise an event, which is received by a mediator. The mediator invokes a BPEL process, which invokes the asynchronous service. A callback is received from the asynchronous service.
Define Defaulting and Derivation Logic

This chapter describes how to define your defaulting and derivation logic, how to use Groovy (a Java-like scripting language), and how to use Oracle Application Development Framework (Oracle ADF) validators and convertor hints instead of using messages.

This chapter includes the following sections:

- Understanding Entity Object Defaulting and Derivation Logic
- About Using Groovy Scripting Language
- Using Validators and Convertor Hints

6.1 Understanding Entity Object Defaulting and Derivation Logic

Defaulting logic means assigning attribute values when a row or entity object is first created or refreshed. (The logic is not re-applied when the entity object is changed.) Defaulting is achieved either declaratively in the default field of the attribute or programmatically by adding code to the EOImpl.

Derivation logic means assigning attribute values when some other attributes have changed. Derivation is achieved either declaratively in the default field of the transient attribute or by using a validator, or programmatically by adding code to the EOImpl.

Figure 6-1, illustrates what you need to consider when determining whether to implement defaulting or derivation logic.
When implementing defaulting or derivation logic, you should also consider the following factors:

- Always assign a valid value to an attribute.
  You should know what the valid values are and there is no reason why you would want to assign invalid values. The end users do not set these values and would have no idea why they would be invalid.
- Always use `initDefaultExpressionAttribute` for calculations that cross containerships. Use `initDefault` for literal or statically computed values.
- Instead of writing code in one of the triggering points during validation or the posting cycle to achieve derivation logic, you can use a method validator or an expression validator.
  When you want the derivation logic to be customizable, the validator approach is preferable. When using this approach the validation result should always be `true` because this is not really a validation logic. You should also make sure that the attribute avoids an infinite loop due to validation.
- You can call either `setAttribute(setter)` or `populateAttribute` to assign the default or derived value to an attribute.
  When you call `setAttribute(setter)`, the logic in the setter is fired and the validation logic is also executed. This does not happen when you call `populateAttribute`.
  In most cases, using `populateAttribute` is sufficient because you should always assign a valid value and therefore do not need to fire validation logic.
However, you may want to call the setter if there is additional logic such as cascading derivation in the setter.

**Tip:**
When you call `setAttribute(setter)` make sure that you do not cause an infinite loop. This may happen due to the attribute and the entity becoming invalid and causing the validation logic to refire.

- You can call `beforeCommit` as well as `setAttribute(setter), validateEntity, and prepareForDML` if your derivation logic involves multiple entities that are not composite.

  For composite object, you can just put your logic either in `validateEntity` or `prepareForDML` of the parent `EOImpl`.

- Oracle ADF handles the propagation of the foreign key ID if there is an association between two entities. This is where the association is defined from the parent entity object to the child entity object, and when the detail entity object is created from the association accessor of the parent entity object. For example:

  ```java
  Row parentRow = ....RowIterator ri = (RowIterator)parentRow.getAttribute("<childEOAccessorName>";
  Row childRow = ri.createRow();
  ```

  Similarly, if there is a view link between two view objects, the framework also handles the foreign key propagation when the child view row is created via the view link accessor of the parent view row.

- List of Values (LOVs) also perform derivation. However, this is at the view object level and you should not place business logic (including derivation) at this level.

  A LOV should only be used on the user interface (UI) to show a list of valid values or as a service to derive the foreign key ID based on the foreign alternate key.

### 6.2 About Using Groovy Scripting Language

ADF Business Components now provide integrated support for Groovy (a Java-like scripting language), which is dynamically compiled and evaluated at run-time. Because it is dynamically compiled, Groovy script can be stored inline in the XML and is eligible for customization. Groovy also supports object access via dot-separated notation, which means you can now use syntax such as `empno` instead of `getAttribute(EMPNO)`.

You can embed Groovy script into various declarative areas, including:

- **Validation** - Use a Groovy script that returns `true` or `false` for declarative validation.
- **Validation Error Messages** - Use Groovy expressions to substitute the tokens in the error message.
- **Bind Variables** - Define the value for a bind variable using a Groovy script expression.
- **View Accessor Bind Variable Values** - Supply bind variable values in a view accessor using Groovy script.
- **Attributes** - Base a transient attribute on a Groovy script. (Currently no UI support).
• **Attribute Default Values** - Define a default value expression on an attribute using Groovy script. (Currently no UI support).

• **Variables** - Define a variable on an entity whose value is computed using Groovy script. (Currently no UI support).

### 6.2.1 About Keywords and Available Names

As with the original Script implementation, the current object is passed into the script as "this" object. Therefore, to refer to any attribute inside the current object simply use the attribute name. For example, in an attribute or validator expression for an entity, to refer to an attribute named `Ename`, the script may say `return Ename`.

There is one top-level reserved name, `adf`, which is used to get to objects that the framework makes available to the Groovy script. Currently, these objects are:

- `ADFContext` (`adf.context`)
- Object on which the expression is being applied (`adf.object`)
- Error handler that lets the validator generate exceptions or warnings (`adf.error`)

All other names come from the context in which the script is applied:

- **Variable** - gets the Variable, `structureDef` in which it is contained via `getStructureDef` method on `VariableImpl`.

- **Transient Attribute** - gets the Entity or ViewRow as its context so that all attributes in the entity are accessible by name. Any method on the entity may be invoked by directly calling the entity method as if you were writing a method in the entity subclass.

  **Tip:**
  
  Only public methods on the entity are available to call.

  You also need to call the method using the "object" keyword, such as `adf.object.createUnqualifiedRowSet()`. The "object" keyword is equivalent to the "this" keyword in Java. Without it, in transient expressions, the method is assumed to exist on the script object itself, which it does not.

- **Validator** - gets the Validator context `JboValidatorContext` merged with the Entity on which the validator is applied. This is done so that you can use:
  - `newValue` and `oldValue` to get to the values being validated
  - `sourceRow` to get to the Entity or ViewRow on which the validator is applied
  - All attribute names in the Entity or ViewRow as top-level names

### 6.2.2 About Scripting Logic

Groovy scripting logic is similar to Expression Language (EL) because you can use a "." separated path to get to a value inside an object. Note that if a Java object implements Map, only the map lookup is performed instead of the bean style property lookup. However, for Maps that extend `JboAbstractMap` you get the same EL behavior, which is map first followed by bean lookup. This is due to the implementation of `get` in `JboAbstractMap`. 
Consider the following information:

- All Java methods, language constructs, and Groovy language constructs are available in the script.

- Aggregates are implemented by calling `sum(expr)`, `count(expr)`, or `avg(expr)` on a `RowSet` object where `expr` can be any Groovy expression that returns a numeric value or number domain.

- The `defaultRowSet` reserved keyword has been removed. The method `EntityImpl.createUnqualifiedRowSet()` replaces `EntityImpl.getDefaultRowSet()` and can be accessed like any other public method in `EntityImpl`.

- Use the return keyword as you would in Java to return a value. That is, unless it is a single-line expression where the return is assumed to be the result of the expression itself. For example, "Sal + Comm" or "Sal > 0".

- Do not use `{}` to surround the entire script because Groovy interprets `{` to be the beginning of a Closure object.

- Any object that implements `oracle.jbo.Row`, `oracle.jbo.RowSet`, or `oracle.jbo.ExprValueSupplier` is wrapped into a Groovy `Expando` object. This is to extend the properties available for those objects to beyond the bean properties and also as a way to avoid introspection for most used names.

### 6.2.3 Groovy Expression Examples

The following are some examples of Groovy.

#### 6.2.3.1 Querying Based on the Current Locale

Instead of using the following SQL to achieve this:

```sql
SELECT C.ISO_COUNTRY_CODE, C.COUNTRY_NAME
FROM COUNTRY_CODES C
WHERE LANGUAGE = SYS_CONTEXT('USERENV', 'LANG')
ORDER BY C.COUNTRY_NAME
```

Create a bind variable and base its default value on the `adf.context.locale.language` expression:

#### 6.2.3.2 Error Message Tokens

To get the attribute new value and label:

```text
Error Message: ProductID_Rule_0

Errors: Product Identifier (prod) is an invalid value for (label).

Error Message Expressions

<table>
<thead>
<tr>
<th>Index</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>prod</td>
<td>newVal</td>
</tr>
<tr>
<td>label</td>
<td>source.ProductID Label</td>
</tr>
</tbody>
</table>
```
The above example uses the following two Groovy expressions:

```
newValue  // This works because an attribute level validator has been created.
source.hints.ProductId.label

and

source.structureDef.name" of type "+sourceFullName
```

### 6.2.3.3 Expression Validators

**Example 6-1** is an example of an Object graph, custom error, and a warning:

**Example 6-2** is an example of how to average a collection.

**Example 6-3** is an example of a built-in or custom method call on the sourceObject of this validator (sourceObject being the Entity on which this validator is being run). isAttributeChanged(String) is a public method on the EntityImpl.

**Example 6-4** is an example of getting to oldValue / newValue of an attribute on which this validator is applied:

**Example 6-5** is an example of accessing the Entity state relative to the database and relative to the last post operation.

Use adf.object.entityState or adf.object.postState.

To get the old value of an attribute (this works in the context of a transient Entity Object attribute):

**Example 6-6** is an example of the WHRxle construct as well as calling an accessor (Emp):

**Example 6-1    Object Graph, Custom Error, and a Warning**

```
if (EmpSal >= 5000)
{
  // If EmpSal is greater than a property value set on the custom
  // properties on the root AM
  // raise a custom exception else raise a custom warning
  if (EmpSal > source.DBTransaction.rootApplicationModule.propertiesMap.salHigh)
  {
    adf.error.raise("ExcGreaterThanApplicationLimit");
  }
  else
  {
    adf.error.warn("WarnGreaterThan5000");
  }
}
else if (EmpSal <= 1000)
{
  adf.error.raise("ExcTooLow");
}
return true;
```

**Example 6-2    Averaging a Collection**

```
attribute Number EmpSal : SAL
|
expressionValidator(expression =
  "newValue <= source.createUnqualifiedRowSet().avg("EmpSal") * 1.2");
```
Example 6-3  Built-in or Custom method Call

```java
if (source.isAttributeChanged("EmpSal") || source.isAttributeChanged("EmpComm")) {
    return true;
} return false;
```

Example 6-4  Getting to Old Value and New Value of an Attribute

```java
return (oldValue == null || newValue < oldValue * 1.2);
```

Example 6-5  Getting the Old Value of a Transient Entity Object Attribute

```java
index = object.getStructureDef().getAttributeIndexOf("Salary");
return object.getAttribute(index, oracle.jbo.server.EntityImpl.ORIGINAL_VERSION);
```

Example 6-6  While Construct and Calling an Accessor

```java
emps = Emp;
boolean alreadyfound = false;
emps.reset();
while (emps.hasNext()) {
    if (emps.next().Job == "CLERK") {
        if (alreadyfound) {
            adfError.raise("alreadyfound");
        }
        alreadyfound = true;
    }
}
return true;
```

6.2.3.4 Attribute Defaulting and Calculation

Example 6-7, Example 6-8, and Example 6-9 are examples of a simple transient attribute, how to sum or count a collection, and how to create a complex calculation of a bind variable value.

Example 6-10 is of an entity-attribute XML fragment where a transient expression is used to provide a default value for that attribute. This expression is evaluated before the protected `create` method of the entity is called. Example 6-11 is an example of an attribute defaulting with a transient attribute calculation expression.

Example 6-7  Simple Transient Attribute

```java
attribute transient Integer YearSal {
    transientexpression = "EmpSal * 12";
}
```

Example 6-8  Sum or Count a Collection

```java
attribute transient Integer TotalSal {
    transientexpression = "object.createUnqualifiedRowSet().sum("EmpSal")";
}
attribute transient Integer TotalCount {
    transientexpression = "object.createUnqualifiedRowSet().count("EmpSal")";
}
```
Example 6-9  Complex Calculation of a Bind Variable Value

```groovy
query EmpView
{
  entity Emp EmpUsage \*;
  where "SAL > :avgSal"
  orderby "1"
  bindingstyle "OracleName"

  variables
  {
    Double avgSal
    kind (where)
    {
      transientexpression
      {
        totSal = 0;
        empCount =0;
        fullVO = structureDef.getApplicationModule().createViewObject("_AvgSal",
          testp.kava.VO7.si33mt.EmpAllView);
        empCount = 0;
        while (fullVO.hasNext())
        {
          row = fullVO.next();
          sal = row.EmpSal; totalSal = totSal + sal; empCount = empCount + 1;
        }
        fullVO.remove();
        if (empCount > 0)
        {
          return (int)(totalSal / empCount);
        }
        else
        {
          return 0;
        }
      }
    }
  }
}
```

Example 6-10  Attribute Value Defaulting

```groovy
<Attribute
  Name="EmpComm"
  ColumnName="COMM"
  Type="oracle.jbo.domain.Number"
  ColumnType="NUMBER"
  SQLType="NUMERIC"
  TableName="EMP">
  <TransientExpression><![CDATA[
    if (EmpSal == null)
    {
      return null;
    }
    if (EmpDeptNum == null)
    {
      return 0;
    }
    if (EmpDeptNum > 40)
    {
      return 500;
    }
  ]]>]]>
</Attribute>
```
Example 6-11    Attribute Defaulting with a Transient Attribute Calculation Expression

```xml
<ViewAttribute
    Name="Total"
    IsUpdateable="false"
    AttrLoad="Each"
    IsSelected="false"
    IsPersistent="false"
    PrecisionRule="false"
    Type="java.lang.String"
    ColumnType="VARCHAR2"
    AliasName="View_ATTR"
    SQLType="VARCHAR">
    <TransientExpression>
        <![CDATA[
        if (Sal != null && Comm != null)
        {
            return Sal + Comm;
        }
        else
        {
            return Sal;
        }
        ]]>
    </TransientExpression>
</ViewAttribute>
```

6.2.4 About Defining Expressions at Design Time

An expression for an attribute can be defined using either the Attribute Editor (see Figure 6-2) or the Expression Editor (see Figure 6-3).

Figure 6-2    Entity Object — Attribute Editor

If you want to define a string literal instead of a Groovy expression, select Literal as the Value Type and enter the value as "My Literal Value".
To access the Expression Editor, click the **Edit** button located next to the **Value** text box.

**Note:**

Recalculate Expression is used to determine whether the expression must be recalculated as changes are made during run-time. The Recalculate option is hidden for persistent attributes. This is because Persistent attribute values are always updateable by the user and therefore, the expression of the attribute should only act as a default expression so recalculation is not necessary. For non-persistent attributes, the user can choose to always recalculate, never recalculate, or decide if recalculation is needed based on the evaluation of the recalculate expression.

6.3 Using Oracle ADF Validators and Convertor Hints

An Oracle ADF validator is appropriate for simple data entry errors that the user can easily correct immediately, on the same UI. In some situations, you should consider using these validators or converter hints instead of using messages.

However, errors such as those that require you to leave the UI to fix, require administrator intervention or an incident report to be filed, should be a Message Dictionary message instead.

**Caution:**

Oracle ADF validators and converter hints can only be used with messages stored in the Strings resource bundles. They cannot be used for messages stored in the Message Dictionary.
To ensure that the user has supplied the correct sort of value or a value in a valid range, input fields can be validated using an Oracle ADF validator. Values may be converted by a converter, for example to convert a string of input characters into a value of some other type such as a date or color.

To validate or convert an input value, you add the input component to the page and then add a validator or converter to that field. Each validator and converter has some messages associated with it:

- **Hints** to display to the user details of what sort of value they need to enter.
- **Error messages** to display if the user enters an invalid value.

For an individual component, you can explain the error to a user in terms relating to that specific input component by overriding the hints or by adding or overriding a detailed error message.

### 6.3.1 Override an Oracle ADF Validator or Converter Message with New Text

You may not see any messages when you follow these steps to select the Application Messages resource bundle:

1. In JDeveloper, select the af validator tag in the UI page.
2. Open the Property Inspector.
3. Select the message attribute.
4. Select the text resource.
5. Select the Application Messages resource bundle.

In this case, you may need to override the default message from the validator. To do so, follow the procedure in "Displaying Hints and Error Messages for Validation and Conversion" in the *Developing Web User Interfaces with Oracle ADF Faces*. 

Define Defaulting and Derivation Logic 6-11
7

Define and Use Message Dictionary Messages

This chapter provides a detailed overview of Message Dictionary messages and discusses how to use them in Oracle Fusion Applications.

This chapter contains the following sections:

- Introduction to Message Dictionary Messages
- Understanding Message Types
- Understanding Message Content
- About Grouping Messages by Category and Severity
- Understanding Incidents and Diagnostic Logs with Message Dictionary
- Using Message Dictionary Messages in Oracle ADF Java Code
- About Associating Message Dictionary Messages with Oracle ADF Validation Rules
- About Raising Error Messages Programmatically in PL/SQL
- About Diagnosing Generic System Error Messages
- About Formatting Message Dictionary Messages for Display in Oracle ADF Applications
- About Integrating Messages Task Flows into

7.1 Introduction to Message Dictionary Messages

The Message Dictionary stores translatable Error and Warning messages for Oracle Fusion Applications. These types of messages provide information about business rule errors, such as missing or incorrect data, and how to resolve them, warn about the consequences of intended actions, inform about the status of an application, page, or business object, and indicate that processes and actions are performing or are completed.

All other messages, except the strings and messages that need to be accessed by C or PL/SQL programs, can be stored in resource bundles (resource bundles only can be accessed by Java programs). The strings and messages that need to be accessed by C or PL/SQL programs can be stored in the Message Dictionary as Information and UI String messages. Resource bundles also can be used to store job output or log file messages for Oracle Enterprise Scheduler (ESS) Java programs and test output messages for Java Diagnostic Testing Framework tests.
The Error, Warning, Information, and UI String message types are described in detail in Understanding Message Types.

**Note:**
Because the messages are stored in Application Object Library FND_MESSAGE_% tables, these types of messages are sometimes referred to as FND messages.

By using the messages in the Message Dictionary, you can define standard messages that you can use in all your applications, provide consistency for messages within and across all your applications, define flexible messages, and change or translate the text of your messages without regenerating or recompiling your application code.

**Note:**
Non-message strings, such as labels, report headings, and message fragments are typically stored in Oracle Application Development Framework (ADF) resource bundles. However, because Oracle ADF resource bundles only can be accessed by Java programs, you can store these types of strings in the Message Dictionary if C or PL/SQL programs need to access them.

Message Dictionary messages are composed of several message components, which enable you to author different messages for different audiences, such as the end user or help desk personnel, and for different conditions, such as when an action must be performed before the user can continue. For more information, see About Message Components.

Messages can be displayed to the UI and written to logs and incidents. Incidents are collections of information about system errors for which end users might require assistance from help desk personnel. An incident contains information about the state of the system at the time the problem occurred. Help desk personnel can use incidents to supply internal support personnel or Oracle support personnel with information about problems that need to be resolved.

You can set up a Message Dictionary message such that an incident and an associated log entry are created automatically. This is referred to as implicit incident creation. For information on how to generate incidents and log entries from Message Dictionary messages, see Understanding Incidents and Diagnostic Logs with Message Dictionary. For information about how incidents and log entries can be used, see the following chapters in the Administrator’s Guide:

- "Troubleshooting Oracle Fusion Applications Using Incidents, Logs, QuickTrace, and Diagnostic Tests"
- "Understanding Oracle Fusion Applications Incidents and Log Files"

### 7.1.1 Creating and Reviewing Messages

Use the Manage Messages task in the application’s Setup and Maintenance work area to create and maintain Message Dictionary messages. For information about implementing your specific product family, do the following:

1. Access the Oracle Fusion Applications library.
2. See the Implementing Common Features guides for your product family.

Oracle Fusion applications provide some common messages in the Message Dictionary with message names that begin with FND_CMN_. Do not modify or replace these messages.

7.2 Understanding Message Types

All messages must have a message type. The message type indicates which message components are applicable, determines whether implicit logging and incident creation occurs, and determines the logging level if the message is logged. For information about message components, see About Message Components. For information about logging and incident creation, see Understanding Incidents and Diagnostic Logs with Message Dictionary. For information about the standard log settings and about logging profile options see the "Default System Log Settings" section in the Administrator’s Guide.

The valid values for message types are fixed and therefore cannot be customized.

**Error Messages**

Use the Error message type for messages that alert the user to data inaccuracies when completing a field, submitting or saving a page, navigating from the page, or when an application or unknown failure occurs. An error message requires attention or correction before the user can continue with their task.

**Warning Messages**

Use the Warning message type for messages that inform users about an application condition or a situation that might require their decision before they can continue. Warning messages describe the reason for the warning and potential consequence of the selected or intended action by users. The warning requires the attention of users, and a standard question might be posed with the warning, or the warning can take the form of a statement.

**Information Messages**

The Information message type is intended for the following types of strings:

- Non-error and non-warning text and messages that are accessed by C or PL/SQL code or are used by Oracle Enterprise Scheduler (ESS) statuses or log files. Because C and PL/SQL programs cannot access Java-based resource bundles, these strings must be stored in the Message Dictionary.

  An example of such a string is the completion text for an ESS program or process.

- Strings that are written to ESS job log or output files, as shown in Figure 7-1. These strings are used to provide a text record about request processing execution, failures, and so on.
Figure 7-1  Processing Request Output Log File

Strings for information level diagnostic logging.

You can choose to use the Information message type in one of two ways:

- Use the Information message type for all non-error and non-warning messages.
- Use the Information message type to store complete messages and use the UI String message type, which is described in the next section, to store fragments. For example, if your messages must pass a review process, you might choose to use the UI String message type for messages that do not need to conform to message guidelines.

UI String Messages

Use the UI String message type for non-error and non-warning strings that need to be stored in the Message Dictionary but are not complete messages, such as prompts, titles, or translated fragments. For example, "Upload Process Parameters." Note that UI String messages are processed exactly as Information messages.

7.3 Understanding Message Content

Messages must have unique message names. Although message numbers are not required, you should use them for error messages to make it easier for users to identify the precise error in logs, and to enable users to find more information about the error in various help sources, including those in different languages.

Translation Notes are required for many messages. You are encouraged to write a complete description that describes the use case for all of your messages.

Different combinations of information are provided depending on the nature of the message and the intended audience, such as end user or help desk personnel. This is accomplished using message components.

Tokens are also an important part of messages. Tokens are the programmatic parts of message text that allow the substitution of other text or values into the message at runtime. They are used as a way to include variable information in the same message. In Oracle Fusion Applications, tokens are used for dates, numbers, and specific types of text.

7.3.1 About Message Names

Every message must have a unique name. You should include a unique prefix that makes it easier to find your custom messages and that helps to avoid name conflicts with non-custom messages. Names that begin with FND_CMN_ are reserved for Oracle Fusion Applications common messages.
7.3.2 About Message Numbers

A unique and persistent message number can be included with each message. The message range 10,000,000 to 10,999,999 has been allocated for customers' own messages.

When displayed, the number takes the format of (Application Shortname-Number). For example:

Descriptive flexfields do not support unit of measure enabled segments. (FND-2774)

If the message does not have a message number, the formatted number is not displayed.

7.3.3 About Translation Notes

A translation note (message context) is a descriptive note to developers, translators, and message customizers describing where and how the message is used. The note is not translated and cannot contain tokens. It is never displayed to end users or help desk personnel. The maximum size of this field is 4000 characters.

7.3.4 About Message Components

Message components enable you to define messages for different audiences and address additional information needs. All messages require a value for the Message Text component, the other components are optional.

Both help desk personnel and end users see the message text and cause components. For the other components, you can use the Message Mode profile option, which has a code of FND_MESSAGE_MODE, to configure whether the end user or help desk personnel (or both) see each type of component. For example, you can set the profile option to enable a particular user to see the Message Admin Detail component. You use the Manage Administrator Profile Values task in the Oracle Fusion Applications Setup and Maintenance work area to set the Message Mode profile option to Administrator or User at the Site, Product, and User levels.

Note:

Incidents contain all message components. For more information about incidents, see Understanding Incidents and Diagnostic Logs with Message Dictionary.

Message Text

Message text is required. This is a brief statement of the operation attempted and the problem that consequently occurred, or information that the user needs to know. The text is included in log and incident creation messages. The content in this field is customizable and the text can contain tokens. The maximum field size for messages stored in the Message Dictionary is 240 characters.

If the entire message, after tokens have been substituted, exceeds the 240 character limit, the message text is truncated. To allow room for expansion in other languages, the US version of any translated column should be no more than 70% of the maximum possible length. (For example, 240 character short text becomes 160 characters in US).
Caution:

Tokens are just values substituted into the message at runtime. Tokens must come from a translated source unless it is a number, seed data, technical information, or a name that is not translated. Extreme care must be taken with tokens when substituting translatable data. You must make sure that it makes sense at run-time.

For more information, see About Tokens.

The text appears in bold at the top of the message window region. In addition, the message text is the only message component displayed in limited real estate UIs, such as pagers and phones. Therefore, the message text should be clear enough to be understood alone when used in this context. This is a required field for all message types.

Message User Detail

This is a more detailed explanation of the problem identified in the short message and its audience is the end user. This field includes the details that are appropriate and meaningful to the end user and should outline exactly what caused the error to occur. For example, in the case of an incident creation error message, this field can be used to provide the user with information about the type of error. The content in this field is customizable and the text can contain numerous tokens. The maximum field size is 4000 characters.

The text appears in normal letters just below the short message. This field is optional.

Message Admin Detail

Message Admin Detail text provides a detailed explanation of the problem identified in the short message. This information is never seen by the end user. This field is for technical details that are not meaningful to an end user. The content in this field is customizable and the text can contain numerous tokens. The maximum field size is 4000 characters.

For errors that require help desk processing, you must use this component, and it must contain information to assist the help desk personnel to resolve the issue, such as the technical background.

Message Cause

The message cause text provides for the end user a concise explanation of why the error occurred. It lists reasons for the failure such as a prerequisite that is not met, incorrect inputs, an anticipated but incorrect action, and so on. The content in this field is customizable and the text can contain numerous tokens. The maximum field size is 4000 characters.

The word Cause (in bold) is prefixed automatically to the beginning of the cause text. This text appears below the user detail, if available. This component is optional and is only applicable for messages of type Error and Warning. The components Cause and User Action are mutually required, meaning if you enter one you must enter both.

Message User Action

This component is for messages that state the action that the user must perform to continue and complete the task. This is intended for and seen by the end user. The content in this field is customizable and the text can contain tokens. The maximum field size is 4000 characters.
The word **Action** (in bold) is prefixed automatically to the beginning of the action text. This text appears below the cause text. This component is optional and is only applicable for messages of type Error and Warning.

**Message Admin Action**

Message Admin Action messages state the action that must be performed to resolve the error condition. This should contain the information that the help desk personnel requires to resolve the error. The content in this component is customizable and the text can contain tokens. The maximum field size is 4000 characters.

The word **Action**: (in bold) is prefixed automatically to the beginning of the action text. This text appears below the cause text and is only applicable for messages of type Error and Warning. This component is only enabled if Cause and User Action are entered. If this is NULL and User Action information is available, then the User Action information is displayed.

### 7.3.5 About Tokens

Tokens are identified in the message text by their use of curly braces and all uppercase letters. The token values are supplied at runtime by the code that raises the message. For example, the following token `{MATURITY_DATE}` is replaced by a date when the user receives the error message on their screen:

"Enter an effective date that is the same as or later than `{MATURITY_DATE}`".

Becomes:

"Enter an effective date that is the same as or later than 25-APR-2010".

---

**Note:**

Tokens have the following requirements:

- They **must** use curly braces and all uppercase letters.
- They generally must be text, not numbers.
- They cannot contain any spaces, but can contain an underscore.

### 7.4 About Grouping Messages by Category and Severity

You can group messages by category and by severity. These groups are used to define logging and incident policies. Otherwise, category and severity have no affect. Category and severity values do not appear in logging entries, incidents, or the UI.

- **Message Category**: This is a more generic attribute that is used to group messages. For example, all the messages that relate to one functionality, such as a concurrent program, can be grouped together into one category.

  This is an optional field, but it must have a value to enable implicit incident creation.

  Message categories are defined by lookups (of type extensible) so that they can be customized by an administrator. The maximum size of this field is 30 characters. The following are seeded values, but you can add more if required.
**Product** - This value refers to issues concerning product functionality, setup and maintenance. Such messages are typically intended for functional administrators or product super users.

**System** - This value refers to issues concerning the system, database, technology stack, and so on. Such messages are typically intended for technical users such as system administrators or database administrators.

**Security** - This value refers to issues concerning permissions, access, compliance, passwords, and so on. Such messages are typically intended for security administrators.

- **Message Severity**: This grouping attribute is not generic and indicates the severity of the message. You must set the severity to *High* to enable implicit incident creation for the message. The following are seeded values, but you can add more if required.
  - *High* - This value can be used for serious messages that completely stop the progress of an important business process or affect a large user community, and require help-desk personnel’s attention.
  - *Medium* - This value can be used for less severe and more isolated messages.
  - *Low* - This value can be used when it is unclear whether the message has a negative impact on end users or business processes.

Valid message severity values are defined by lookups (of type extensible) so that they can be customized by an administrator. The maximum size of this field is 30 characters.

### 7.5 Understanding Incidents and Diagnostic Logs with Message Dictionary

Incidents are collections of information about system errors for which the customer might require assistance from help desk personnel. An incident contains information about the state of the system at the time the problem occurred. Help desk personnel can monitor and respond to incidents and send them to Oracle if further assistance is necessary. For information about how incidents and log entries can be used, see the following chapters in the *Administrator’s Guide*:

- "Troubleshooting Oracle Fusion Applications Using Incidents, Logs, QuickTrace, and Diagnostic Tests"
- "Understanding Oracle Fusion Applications Incidents and Log Files"

Implicit incident creation and logging occurs when the Message Dictionary message is retrieved in PL/SQL and C code, or when it is formatted in Java code, and the message has the following settings:

- Logging enabled (*loggable_alertable*): *Y*
- Message type: *ERROR*
- Message Category: not null
- Message severity: For incident creation, it must be *HIGH*. For logging, it must be not null.
Note:

Implicit incident creation occurs when logging is enabled. Implicit logging only occurs if the SEVERE log level or more logging are enabled.

Use the Message Dictionary APIs to retrieve a Message Dictionary message. The PL/SQL methods are in the FND_MESSAGE package and the Java methods are in the messageService package. For C code, use the methods in the fddutl package.


For Java code, implicit incident creation and logging occurs for qualifying messages when the appropriate formatting methods are called from the MessageServiceAMImpl class and the MessageServiceAM interface, such as the getUserXML(), formatMap(), formatUserTextMap(), or formatAdminTextMap() methods. See the MessageServiceAM and MessageServiceImpl Javadoc for information about which methods to call for implicit logging. Implicit incident creation and logging also occur when exceptions are created using the ApplcoreException classes.

When implicit logging and incident creation occurs, additional information is appended to the message, as follows:

- If the incident is created in the middle tier using Java, a Business Process Execution Language (BPEL) process, or C, the following note is appended:
  
  An application error has occurred. Your help desk was notified. For more information your help desk may refer to incident {incident number}, {application server name}, {application server domain name}.

- If the incident is created in the database tier using PL/SQL, the following note is appended:

  An application error has occurred. Your help desk was notified. For more information your help desk may refer to incident {incident number_SID}, {database server name}, {database instance name}.

To learn more about the information that is included with incidents and associated log entries, see the How the Diagnostic Framework Works section in the Administering Oracle Fusion Middleware.

### 7.6 Using Message Dictionary Messages in Oracle ADF Java Code

You can use messages in the Message Dictionary in Java code to raise exceptions using Oracle Fusion Middleware Extensions for Applications exception classes. You can also retrieve the message text programmatically.

#### 7.6.1 About Raising Exceptions Using Oracle Fusion Middleware Extensions for Applications Exception Classes

Exceptions from messages in the Message Dictionary should be raised using wrapper classes that are provided in the oracle.apps.fnd.applcore.message package. Wrappers that are provided correspond to the most commonly used Oracle ADF exception classes. See Table 7-1.
Table 7-1  Oracle ADF Exception Classes vs. Message Dictionary Classes

<table>
<thead>
<tr>
<th>Exception Class</th>
<th>Message Dictionary Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>JboException</td>
<td>oracle.apps.fnd.applcore.messages.ApplcoreException</td>
</tr>
<tr>
<td>RowValException</td>
<td>oracle.apps.fnd.applcore.messages.ApplcoreRowValException</td>
</tr>
<tr>
<td>AttrValException</td>
<td>oracle.apps.fnd.applcore.messages.ApplcoreAttrValException</td>
</tr>
</tbody>
</table>

In each of these classes, the message name is expected to be passed in the format: 
APP_NAME:::MESSAGE_NAME (application short name, followed by exactly 3 colons, followed by the message name). For example: "FND:::FND_CMN_POSITIVE".

Message tokens passed to most Message Dictionary Java APIs are expected to be supplied as Map<String, Object> or as an array of alternating String/Object pairs. With either style, the String is the name of the message token and the following Object is an object representing the value of that token. The type of the Object is expected to match the type of the token as shown in Table 7-2.

Table 7-2  Message Tokens and Data Types

<table>
<thead>
<tr>
<th>Token Type</th>
<th>Token Value Object Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXT</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>NUMBER</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>DATE</td>
<td>java.sql.Timestamp</td>
</tr>
</tbody>
</table>

Exceptions that are raised using JboException or one of its subclasses with a severity level of SEVERITY_ERROR, which is the default, or any java.lang.RuntimeException, are treated as system errors, and the following occurs:

- The error message is replaced with a generic message, such as "An application error occurred. Your help desk was informed"
- An incident is created for the system error
- A stack trace is written to the log file for the system error

If you do not want the JboException to be treated as a system error, do one of the following:

- Convert the exception type to ApplcoreException
- Set the severity level to other than SEVERITY_ERROR, such as SEVERITY_RECOVERABLE_ERROR.

Tip:
If you need to see the original error message, you can run the application with the -DAFERROR_MODE=debug parameter, as described in About Diagnosing Generic System Error Messages.
You should use the wrappers wherever possible. However, it is possible to also use
native Oracle ADF exceptions directly if there isn’t a wrapper that exactly suits your
needs. If you do this, you must specify the FndMapResourceBundle resource bundle
class, and format tokens correctly.

Example 7-1 shows sample code that raises an ApplcoreException exception.
Example 7-2 shows an example of raising ApplcoreRowValException exception.
Use of the ApplcoreAttrValException exception is shown in Example 7-3.
Example 7-4 illustrates how to throw a native JBOException.

To display more than one application error message, such as a series of validation
error messages, bundle the exceptions and throw the bundled exceptions, as shown in
Example 7-5. The exceptions in the bundle must be only application error exceptions,
such as ApplcoreException, JboException with
SEVERITY_RECOVERABLE_ERROR, or ValidationException. If you include any
system error exceptions in the bundle, such as NullPointerException, the bundle
is processed as a system error. For example, if you include a RuntimeException
exception in the bundle, you cannot display the bundled exception error messages in a
popup dialog, because it will be processed as a system error.

Example 7-1 ApplcoreException

```java
import oracle.apps.fnd.applcore.messages.ApplcoreException;

// Construct and populate HashMap with token values
Map<String, Object> tokens = new HashMap<String, Object>();
tokens.put("TEXT_TOKEN", "text token value");
tokens.put("NUMBER_TOKEN", new BigDecimal(10));
Calendar cal = Calendar.getInstance();
cal.set(1999, Calendar.DECEMBER, 31, 0, 0, 0);
tokens.put("DATE_TOKEN", new Timestamp(cal.getTimeInMillis()));
throw new ApplcoreException("MYAPP:::MY_MESSAGE_NAME", tokens);
```

Example 7-2 ApplcoreRowValException

```java
import oracle.apps.fnd.applcore.messages.ApplcoreRowValException;

// Construct and populate HashMap with token values
Map<String, Object> tokens = new HashMap<String, Object>();
tokens.put("TEXT_TOKEN", "text token value");
tokens.put("NUMBER_TOKEN", new BigDecimal(10));
Calendar cal = Calendar.getInstance();
cal.set(1999, Calendar.DECEMBER, 31, 0, 0, 0);
tokens.put("DATE_TOKEN", new Timestamp(cal.getTimeInMillis()));
Key key = new Key(new Object[]{"Primary Key"});
ApplcoreRowValException ex = new ApplcoreRowValException("MYAPP:::MY_MESSAGE_NAME", "MyEODefName", key, tokens);
```

Example 7-3 ApplcoreAttrValException

```java
import oracle.apps.fnd.applcore.messages.ApplcoreAttrValException;

// Construct and populate HashMap with token values
Map<String, Object> tokens = new HashMap<String, Object>();
tokens.put("TEXT_TOKEN", "text token value");
tokens.put("NUMBER_TOKEN", new BigDecimal(10));
Calendar cal = Calendar.getInstance();
cal.set(1999, Calendar.DECEMBER, 31, 0, 0, 0);
```
Example 7-4  Native JBOException

```java
import oracle.apps.fnd.applcore.messages.ApplcoreException;

// Construct and populate HashMap with token values
Map<String, Object> tokens = new HashMap<String, Object>;
tokens.put("TEXT_TOKEN", "text token value");
tokens.put("NUMBER_TOKEN", new BigDecimal(10));
Calendar cal = Calendar.getInstance();
cal.set(1999, Calendar.DECEMBER, 31, 0, 0, 0); tokens.put("DATE_TOKEN", new Timestamp(cal.getTimeInMillis()));

JboException ex = new JboException(FndMessagesUtil.getFndMapResourceBundleDef(),
"MYAPP:::MY_MESSAGE_NAME", null);
ex.setErrorParametersMap(tokens);
throw ex;
```

Example 7-5  Throwing Bundled Exceptions

```java
ApplcoreException error1 =
new ApplcoreException("PON:::PON_NEG_REQ_TOTAL_WEIGHT_ERROR");
ApplcoreException error2 =
new ApplcoreException("PON:::PON_KNOCKOUT_CRITERIA_ERROR");

ApplcoreException[] details = {error1, error2};
ValidationException errorBundle = new ValidationException("");
errorBundle.setExceptions(details);
throw errorBundle;
```

7.6.2 About Retrieving Message Text Programmatically

You can use static methods in the `oracle.apps.fnd.applcore.messages.Message` class to retrieve translated, token substituted message text without raising exceptions. APIs are provided to retrieve the fully formatted text of the user message, the administrator message, or to retrieve the parts of the message (short message, cause, action, and so on) individually, as shown in Example 7-6.

Example 7-6  Retrieving Messages

```java
// Construct and populate HashMap with token values
Map<String, Object> tokens = new HashMap<String, Object>();
tokens.put("TEXT_TOKEN", "text token value");
tokens.put("NUMBER_TOKEN", new BigDecimal(10));
Calendar cal = Calendar.getInstance();
cal.set(1999, Calendar.DECEMBER, 31, 0, 0, 0); tokens.put("DATE_TOKEN", new Timestamp(cal.getTimeInMillis()));

// Get the token substituted message short text.
String shortText = Message.getShortText("MYAPP", "MY_MESSAGE", tokens);

// Get the token substituted full user message, in plain text format.
String userText = Message.getUserText("MYAPP", "MY_MESSAGE", tokens);

// Get the token substituted full user message, in HTML format.
String userHTML = Message.getUserHTML("MYAPP", "MY_MESSAGE", tokens);
```
7.7 About Associating Message Dictionary Messages with Oracle ADF Validation Rules

The easiest way to create and manage validation rules is through declarative validation rules. Declarative validation rules are defined using the overview editor for the entity object, and when created, are stored in the entity object’s XML file. These are known as declarative validation rules on entity objects.

For information about defining validation rules on entity objects, see the Defining Validation and Business Rules Declaratively chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

Oracle ADF provides built-in declarative validation rules that satisfy many of your needs. You can also base validation on a Groovy expression, as described in About Using Groovy Scripting Language.

When you add a validation rule, you supply an appropriate error message. You can also define how validation is triggered and set the severity level.

These messages can contain named message tokens for retrieving and displaying context sensitive values.

Tip:

When raising exceptions with the ADF Business Components validation rules, the tokens must be formatted as \{TOKEN_NAME\} and not (TOKEN_NAME).

7.7.1 How to Associate Error Messages with Oracle ADF Entity Object Validation Rules

To associate an error message with your validation rule, do the following:

1. Go to the Failure Handling tab of your declarative validation rule when you have finished defining your rule. In the Validation Failure Severity field, Select Error.

2. Click Select Message to open the Select Text Resource dialog. Choose Application Messages from the Resource Picker dropdown list.

3. Use the Search area to filter your search results. For example, enter FND_VIEW in the search text area to filter your results to messages whose key begins with FND_VIEW.

Note:

You can search for messages only by the message key. All other types of searches have been disabled. Also notice from the results that message keys are prepended with the application short name.

4. Select the required error message from the list of results. The Select Text Resource dialog closes and the selected error message displays in the Failure Message Text area on the Failure Handling tab.
5. If your message contains tokens, bind them to Groovy expressions to retrieve context sensitive values. Groovy script is a Java-like scripting language. For more information about Groovy script, see About Using Groovy Scripting Language.

A validation rule's error message can contain embedded expressions that are resolved by the server at runtime. To access this feature, simply enter a named token delimited by curly braces (for example, \{TOKEN_NAME\} or \{ERRORPARAM\}) in the error message text where you want the result of the Groovy expression to appear.

The Token Message Expressions table at the bottom of the dialog displays a row that allows you to enter a Groovy expression for the token. Figure 7-2 shows the failure message for a validation rule in the PaymentOptionEO entity object that contains message tokens.

![Figure 7-2 Using Message Tokens in a Failure Message](image)

Declarative validation is different from programmatic validation, which is stored in an entity object's Java file. For more information about programmatic validation, see the Implementing Validation and Business Rules Programatically chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.
7.8 About Raising Error Messages Programmatically in PL/SQL

Because they make calls to the database, both PL/SQL and C code require that the message is stored in the Message Dictionary. PL/SQL and C code cannot reference Java-based resource bundles. You can use PL/SQL to:

- Raise exceptions in PL/SQL programmatically using messages from the Message Dictionary
- Retrieve those errors when PL/SQL is called from Java
- Retrieve the message from the stack

7.8.1 About Raising Exceptions Programmatically in PL/SQL

There are three packages that you can use to handle errors in PL/SQL using messages in the Message Dictionary:

- **FND_MESSAGE**: This package includes basic APIs to set messages on the error stack, set tokens, retrieve token substituted message text, and so on.
- **APP_EXCEPTION**: This package includes utilities to raise SQL exceptions with messages in the Message Dictionary as the exception text.
- **FND_MSG_PUB**: This package includes utilities to set messages on the error stack. In Oracle Fusion Applications, the error stack also exists natively in the FND_MESSAGE package; this package is primarily used for backward compatibility with existing code. PL/SQL code that in EBS was primarily called from Framework usually uses this method.

For more information about these packages, see the package headers.

7.8.2 About Raising Errors in PL/SQL

The **FND_MESSAGE** PL/SQL package allows you to set one message and its tokens, as shown in Example 7-7. It also allows you to set multiple messages in the stack by explicitly pushing the current message onto the stack, as shown in Example 7-8. When you need to retrieve the message from the stack, an explicit `pop()` is required, as shown in Example 7-9.

**Example 7-7  Getting Message and its Tokens**

```sql
-- setting INVALID_USER as the current message
fnd_message.set.name('FND', 'INVALID_USER');
-- setting token value for NAME
fnd_message.set_token('NAME', '<USER>');
..........
-- get the current translated and token substituted message
-- then clear the message
msg := fnd_message.get;
```

**Example 7-8  Receiving a Message Record and Clearing Message**

```sql
-- setting INVALID_USER as the current message
fnd_message.set.name('FND', 'INVALID_USER');
-- setting token value for NAME
fnd_message.set_token('NAME', '<USER>');
..........
-- receive a message record which contains everything about the
```
-- current message. The record contains message number, message category
-- message severity and translated and token substituted message text, 
-- translated and token substituted user message, user action, ...
-- Getting message record for current message will NOT clear the message
msg_rec := fnd_message.get_message_record;
-- clear the message
fnd_message.clear;

Example 7-9    Retrieving message from the Stack

-- setting INVALID_USER as the current message
fnd_message.set_name('FND', 'INVALID_USER');
-- setting value for token NAME for INVALID_USER message
fnd_message.set_token('NAME', 'TESTUSER');
-- saving the current message onto stack
fnd_message.push;
-- setting LOGIN_FAILED as the current message
fnd_message.set_name('FND', 'LOGIN_FAILED');
-- saving the current message onto stack
fnd_message.push;
.......
-- poping one message out of stack and set it as the current message
fnd_message.pop;
-- get the translated and token subsituted LOGIN_FAILED message
-- then clear the current message
msg := fnd_message.get;
-- poping one message out of stack and set it as the current message
fnd_message.pop;
-- get the translated and token subsituted INVALID_USER message
-- then clear the message
msg := fnd_message.get;

7.8.3 About Retrieving Errors when PL/SQL is Called from Java

You can use the OAExceptionUtil.CheckErrors() API to check for error messages after calling PL/SQL from Java. The CheckErrors() API looks for errors on both the new FND_MESSAGE and FND_MSG_PUB stacks, raises a bundled exception for each error found on both stacks, and then clears both PL/SQL error stacks.

Where the call to OAExceptionUtil.CheckErrors() depends on which style of error handling your PL/SQL code uses:

- If your PL/SQL code uses FND_MESSAGE with FND_MSG_PUB, then errors will be left on the PL/SQL error stack without raising any exceptions. The call to OAExceptionUtil should go immediately after the PL/SQL call.

- If your PL/SQL code uses APP_EXCEPTION.RAISE_EXCEPTION, or FND_MESSAGE.RAISE, then errors will cause SQL exceptions to be raised. The call to OAExceptionUtil.CheckErrors() should be in a SQLException catch block.

- If you do not know what style of error handling your PL/SQL code uses, or there could be a mixture of both, then you should include calls to OAExceptionUtils.CheckError() in both places, as shown in Example 7-10.

Example 7-10    Calls to OAExceptionUtils.CheckError() — Unknown Error Handling Style

import oracle.apps.fnd.applcore.common.OAExceptionUtils;
...

7-16 Oracle Fusion Applications Developer's Guide
7.9 About Diagnosing Generic System Error Messages

If you see an error message similar to one the following messages, it is because a system error was raised and the original error message was replaced with a generic message:

An application error occurred. Your help desk was informed.

An application error occurred. See the incident log for more information.

When you receive these types of errors, you can look at the log file entry to find the original error message.

Note:

When generic errors are raised, you will see oracle.apps.fnd.applcore.messages.ExceptionHandlerUtil class information at the top of the call stack. This is the code that is replaced the unhandled exception with the generic error and should not be mistaken for the original error from the Message Dictionary.

You can also set one of the following debug options to allow you to see the error more directly, without having to view the log file entry:

- -DAFERROR_MODE=debug: Causes the original error to be displayed in the UI
- -DAFLOG_ECHOED=true: Sends logging output to the console, as well as the log file

For information about finding the cause of an error and its corrective action and for information about viewing and managing log files, see the following chapters in the Administrator’s Guide:
7.10 About Formatting Message Dictionary Messages for Display in Oracle ADF Applications

When raising an exception or attribute validation error by retrieving a message from the Message Dictionary using a resource bundle interface, the exception message returns in XML format.

You can convert XML formatted messages to HTML or plain text for display in Oracle ADF applications, as shown in Figure 7-3.

Figure 7-3  Error Message Example

This can be done in one of two ways:

- Programmatically
- By configuring the error format handler in the DataBindings.cpx file

7.10.1 About Programmatically Converting XML Messages

When directly handling Oracle Fusion Applications resource bundle exceptions in Java code, you can convert XML messages to HTML or plain text using utility APIs. The utility APIs are found in oracle.apps.fnd.appcore.messages.model.util.Util.

Sample code is shown in Example 7-11.

Example 7-11  Converting XML Messages to HTML or Plain Text

```java
Exception exception = new ApplcoreException("FND:::MY_TEST_MESSAGE");

// Retrieve the HTML short message
String htmlShort = Util.formatHTMLMessage(exception);

// Retrieve the HTML message details.
String htmlDetails = (Util.formatHTMLDetailMessage(exception)).getHTMLText();

// Retrieve the plain text message details
String textDetails = (Util.formatHTMLDetailMessage(exception)).getText();

// Retrieve the full plain text message
String textMsg = Util.formatTextMessage(exception);
```
7.10.2 How to Convert XML Messages by Configuring the Error Format Handler

You can convert XML messages to HTML or plain text by configuring the error format handler in the `DataBindings.cpx` file.

To convert XML messages to HTML, do the following:

1. Under the user interface project, open the `DataBindings.cpx` file.

   Tip:
   JDeveloper names the user interface project `ViewController` by default.

2. In the Property Inspector, set the **ErrorHandlerClass** field to the value shown in Example 7-12 and Figure 7-4.

   **Figure 7-4** Setting the Value of the **ErrorHandlerClass** Field in the Property Inspector

   ![Property Inspector screenshot]

   **Example 7-12** The Value of the **ErrorHandlerClass** Field

   `oracle.apps.fnd.applcore.messages.MessageFormatHandler`

7.11 About Integrating Messages Task Flows into Oracle Fusion Functional Setup Manager

Every Oracle Fusion application registers task flows with a product called Oracle Fusion Functional Setup Manager. These task flows are available from the application’s Setup and Maintenance work area and enable customers and implementers to set up and configure business processes and products.

For information about implementing your specific product family, do the following:

1. Access the `Oracle Fusion Applications` library.

2. See the Implementing Common Features guides for your product family.

Function Security controls your privileges to a specific task flow, and users who do not have the required privilege cannot view the task flow. For more information about
how to implement function security privileges and roles, see Implementing Function Security.

Table 7-3 lists the task flows and their parameters.

**Table 7-3  Messages Task Flows and Parameters**

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Parameters Passed</th>
<th>Behavior</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Messages</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/messages/ui/flow/ManageMessagesTF.xm#ManageMessagesTF</td>
<td>mode='search' [moduleType] [moduleName] [applicationId] [pageTitle]</td>
<td>Search mode launches the search page, with optional parameters to restrict to a particular module.</td>
<td>NA.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode='edit' messageName applicationId [pageTitle]</td>
<td>Edit mode launches the edit page for a particular message. The messageName and applicationId parameters are mandatory as they specify the message to edit.</td>
<td></td>
</tr>
</tbody>
</table>

About Integrating Messages Task Flows into Oracle Fusion Functional Setup Manager
This chapter describes how to share reference data across organizations by using setIDs to partition the data into different sets of values. Each organization can then maintain its data in a common table, using a set of values specific to that organization.

This chapter includes the following sections:

- Introduction to SetIDs
- About Implementing SetID on Entity Objects
- About Integrating SetID Task Flows into

8.1 Introduction to SetIDs

Different organizations within a single company often need to use different sets of reference data to serve the same purpose. For example, the job codes for one country might be different from the job codes for another country. Different Oracle Fusion Applications customers should be able to make their own decisions about how to define the job codes, and be able to define a separate set for each organizational section of the enterprise. They should also be able to define a common set or sets and instruct the system which set is required by which organizations. For example, job codes for software engineers might be MTS, SMTS, PMTS; job codes for managers might be M1, M2, M3. SetIDs enable them to accomplish this easily.

For information about set-enabling lookups, see Implementing Lookups.

SetID Implementation

After you have completed the development process as discussed in this chapter, and delivered your application with the ability to use set-enabled reference data, application implementers and administrators must then be able to define and maintain reference data sets and set assignments that are appropriate to the organization which will use the application. They can accomplish these tasks using the Manage Reference Data Sets and Manage Reference Data Set Assignments applications, respectively.

You make these setup applications available to implementers and administrators by incorporating their task flows into Oracle Functional Setup Manager. For more information, see About Integrating SetID Task Flows into.

For information about implementing your specific product family, do the following:

1. Access the Oracle Fusion Applications library.
2. See the Implementing Common Features guides for your product family.
8.1.1 About Partitioning by SetID

SetIDs enable you to share a set of reference data across many organizations. Sharing reference data is a method of limiting the set of available values to those that are appropriate for a validated attribute. Some benefits of this include:

- The list of values for a field in a user interface is reduced
- An attribute passed into an API is validated against the limited set of values

The end goal is to save customers some effort in maintaining reference data by enabling it to be shared between different parts of the organization that implements applications. Reference data should not need to be maintained in multiple places at multiple times. Reference data is data in tables that you do not regard as transactional and high volume; for example, payment terms that can be used on a customer invoice.

By dividing the reference data into partitions appropriate to the organizational entities that will use the data, setIDs enable you to share control table information and processing options among business units. The goal is to minimize redundant data and system maintenance tasks. For example, you can define a group of common job codes that are shared between several business units. Each business unit that shares the job codes is assigned the same setID for that record group.

SetIDs can be thought of as a striping technology to partition referenced data. All shared reference tables can be striped with a setID column to enable partitions (or sets). This does not require you to change the tables’ primary keys.

With partitioning, a customer can choose to have reference data sets specific to each organizational unit mapped one-to-one, or have several different organizational units use the same set of reference data. Customers, rather than development, will have the choice in determining what level of sharing or exclusivity they would like to maintain in the reference data.

A setID is the means by which applications can filter reference data into subsets when they are referenced by different transactional entities. The filtering is driven, indirectly, by contextual values available in the referring transactional entity.

8.1.2 About SetID Determinant Types

Use of the shared data partitions is facilitated by a context setting called the determinant, which is usually a column on the referring transactional entity. The purpose of the determinant is to identify an organizational subset; you use it to specify which reference data is valid for use in a given business context. The determinant is the value of a transactional column that is one of several designated determinant types. If at least one column of the transactional table is a setID determinant type, data sharing may make sense for the transaction.

For example, different business units may use the same office supply vendor, but have different requirements for which supplies can be purchased. The determinant type and value provide part of the criteria for selecting the appropriate office supply reference data set.

In addition to the presence of a determinant on the transactional entity, the data that you want to reference must be set-enabled as described later in this documentation.

The setID determinant type can be one of the following existing fields:

- **Asset Book** — A book that contains assets belonging to a business unit or ledger. It holds information about the asset’s acquisition, depreciation, and retirement. An
asset may be assigned to one or more books; for example, the corporate, tax and budget books.

- **Business Unit** — This roughly corresponds to a department or organization. For example, Virgin might have an airline, a store, and a recording label as different business units.

- **Cost Organization** — A cost organization groups inventory organizations within a legal entity to achieve the following:
  - Establish the cost accounting policies for the inventory organizations.
  - Support cost accounting reporting.
  - Allow the definition of defaults.
  - Allow multiple inventory organizations to share cost calculation.
  - Restrict role access to costing data.

Cost organization will likely map into a company's enterprise structure as a cost department.

- **Project Unit** — A logical organization within a company created to ensure and enforce consistent project management practices.

- **Reference Data Set** — For cases where shared reference data has references to other shared reference entities.

### Some Criteria for Selecting a Determinant Type

To help decide what determinant type to use for a given application, consider the following:

- If you cannot change the reference data for different parts of a deploying enterprise, the reference data is global and partitioning is not required.

  Examples of data suitable for partitioning include (but are not limited to) units of measure, currency codes, country codes, or anything else governed by a standard.

- If the values for the reference data will be decided by the general manager, the best reference data set determinant is likely to be the **Business Unit**.

For more information about setID determinant types, see [How to Annotate Transactional Entity Objects for SetID](#).

### 8.1.3 Understanding SetID Machinery

SetID Machinery is the collection of Applications Core Technology software elements that act in concert to facilitate the use of setIDs to partition, access and maintain reference data. At a high level, the machinery is comprised of:

- SetID configuration tables
- SetID metadata for business objects and extensions to ADF Business Components middleware
- SetID design-time extensions
- SetID summary tables
The following sections introduce the elements of setID machinery and the ways in which they can be used to implement data sharing.

### 8.1.3.1 About Partitioning Patterns

There are three setID partitioning patterns. Choose one of these patterns based on your business requirements:

- **Row striping** *(ROWSTRIPE)* — This is the simplest pattern, and the default. In this pattern the SET_ID column is just a striping column, and is not part of the set of unique keys for the table. You can filter as follows:

  ```sql
  WHERE SET_ID = :1
  ```

- **Row striping with common rows** *(COMMON)* — This is the same as the row striping pattern, with the addition of a COMMON partition. You filter as follows:

  ```sql
  WHERE SET_ID IN (:1, 0)
  ```

  **Note:**
  The set with setID of 0 is seeded as the common set. This set will be available for assignment only if you select **Row Striping With Common Rows** for the reference entity.

- **SetID subscription** *(SUBSCRIPTION)* — The drawback of the first two patterns is that if reference data must be in two different partitions (other than the common one), it has to be copied and placed in both sets. To avoid that, a setID subscription table can be introduced and used to list which sets include each row. This will allow the same reference data to be in two different sets without the need to copy the data for each set. You join your reference entity with the setID subscription table and filter as follows:

  ```sql
  SET_ID: WHERE PARENT.PK1=SUBSCRIPTION.PK1 AND SUBSCRIPTION.SET_ID = :1
  ```

For more information about partitioning patterns, see [How to Annotate Reference Entity Objects for Sharing](#).

### 8.1.3.2 About Reference Groups

In addition to tables, other sources of reference data such as lookup types and views can also be shared using setIDs. These are all generically referred to as **reference entities**. Reference entities are generally considered to be setup data, and they may be implementing business policies and legal rules. Reference entities in your application are grouped into logical units called **reference groups**, based on the functional area and the partitioning requirements they have in common. For example, all tables and views that define Sales Order Type details might be part of the same reference group.

Figure 8-1 illustrates an example of a Worker Assignment transaction table with two set-enabled references: a reference to Salary Codes with partitions determined by Business Unit, and another reference to Labor Agreement with partitions determined by Cost Organization.
8.1.3.3 About Set Configuration Tables

There are five types of set configuration tables:

- Sets
- Reference groups
- SetID assignments
- Reference entities

Sets Table

The sets table, FND_SETID_SETS, lists all of the sets defined for Oracle Applications, plus any new sets that you define. It includes the columns SET_ID and SET_NAME, which enable you to select the proper SET_CODE.

Sets listed in this table include:

- The two default seeded sets, COMMON and ENTERPRISE.
- Default sets that map to existing transaction data, created as an upgrade to Applications Unlimited.
- New sets created by customers, to implement set-enabled reference entities specific to their organizations.

Reference Groups Table

The reference groups table, FND_SETID_REFERENCE_GROUPS_B, captures the default determinant type for all reference entities in each group. This table uses the primary key of REFERENCE_GROUP_NAME. It also includes the APPLICATION_ID column, which is used for filtering and managing ownership.
The available reference groups defined in the reference groups table will be populated before you start creating entity objects. Reference group definitions are owned by the application that owns the reference entities in that group. Application development teams are ultimately responsible for defining and delivering reference groups.

**Note:**

Only reference entities that might be referenced as setID targets need to be captured here; this is not intended to be an exhaustive inventory of all tables in the applications. For more information about reference groups, see About Reference Groups.

---

**SetID Assignments Table**

A transactional entity may have multiple sets of reference data that are treated in the same manner. For this reason, reference data sets are assigned to a reference group, then the setID assignment is configured for each determinant value, determinant type, and reference group.

The setID assignments table, FND_SETID_ASSIGNMENTS, records which set to use in every reference table for every determinant value. It is a SQL-joinable entity that can be used to convert available context information into a setIDentifier suitable for filtering rows from referenced entities. The context information serves as the table’s primary keys:

- REFERENCE_GROUP_NAME
- DETERMINANT_TYPE
- DETERMINANT_VALUE

Based on these keys, you can determine a setID.

**Note:**

Although development may seed this table with default values, it will be accessed by customers to implement set-enabled reference entities specific to their organizations.

---

**Reference Entities Table**

The reference entities table, FND_SETID_REFERENCE_ENTITIES, contains the list of all setID enabled non-lookup reference entities. The SET_ID_PATTERN column indicates which setID pattern is being used by each reference entity. If the value of this field is SUBSCRIBE (setID subscription), the column SET_ID_CHILD_TABLE will be populated with the setID subscription table name.

**Note:**

For customers, this table is read-only.

---

**8.1.3.4 About SetID PL/SQL Utilities**

The setID PL/SQL utilities are APIs that include the following packages:
Fnd_setid_sets_pkg package
This package contains table handlers for fnd_setid_sets table.

Fnd_setid_assignments package
This package contains table handlers for fnd_setid_assignments table.

Fnd_setid_reference_groups package
This package contains table handlers for fnd_setid_reference_groups table.

Fnd_setid_ref_entities_pkg package
This package contains table handlers for fnd_setid_reference_entities table.

Fnd_setid_set_groups package
This package contains table handlers for fnd_setid_set_groups and fnd_setid_set_group_members tables.

Fnd_setid_utility package
This package contains the following utilities:

• isValid

```java
/**
 * Returns true if the given parameters are valid, false if not.
 * @param referenceGroupName The reference group name
 * @param setIdDeterminantType The determinant type which could be: BU, RR, LE...
 * @param setIdDeterminantValue The determinant value.
 * @param setId The setid value.
 * @return true or false.
 */
function isValid(X_REFERENCE_GROUP_NAME in varchar2,
                 X_DETERMINANT_TYPE in varchar2,
                 X_DETERMINANT_VALUE in varchar2,
                 X_SET_ID in number) return boolean;
```

• getSetId

```java
/**
 * Returns the set ID corresponding to the specified determinant value, type,
 * and reference group name. This method implements an LRU cache to speed
 * up the lookup.
 * @param setIdDeterminantValue The determinant value
 * @param setIdDeterminantType The determinant type which could be: BU, RR, LE...
 * @param referenceGroupName the reference group name.
 * @return the corresponding set ID value.
 */
function getSetId(X_REFERENCE_GROUP_NAME in varchar2,
                  X_DETERMINANT_TYPE in varchar2,
                  X_DETERMINANT_VALUE in varchar2) return number;
```

• getReferenceGroupName

```java
/**
 * Returns the reference group name based on a given reference entity name.
 * @param referenceEntityName The name of the table to obtain
 * the reference group for
 */
```
8.2 About Implementing SetID on Entity Objects

You define the following information to implement shared (that is, set-enabled) reference entities:

- On the reference entities to be shared:
  - You provide the shared reference entity group name.
  - You specify which setID pattern to use: row striping, row striping with common rows, or setID subscription.
  - In the case of the setID subscription pattern, you specify the subscription table name.
  - You make sure the attribute corresponding to setID is named SetId, and specify the determinant type as SET.
  - You make sure that any view objects built on the reference entity include the SetId attribute.

For more information about setID partitioning patterns, see About Partitioning Patterns.

- On the transactional entities that will use the shared reference data:
  - You build entity associations to all shared reference entities.
  - You specify which attributes are determinants by specifying the determinant type.
  - On foreign keys that point to shared reference entities, you indicate which determinant attribute drives the set of that reference entity.

Before you begin:

Following are the activities that you should complete before you engage in set-enabling references or lookups:

- Determine which reference entities you want to partition for sharing, and set-enable them by adding a SET_ID column.

- Generate ADF Business Components entity objects for your set-enabled reference entities and transactional entities. Make sure that your entity objects extend from
Oracle Applications base classes (if available) under
\texttt{oracle.apps.fnd.applcore.oaext.model}.

For more information, see the "Creating a Business Domain Layer Using Entity Objects" chapter in the \textit{Developing Fusion Web Applications with Oracle Application Development Framework}.

- Generate ADF Business Components entity associations for all of your entity objects.

  The following setID metadata will be saved in ADF Business Components metadata as properties:
  - The reference group name of reference entities.
  - The setID pattern used by each shared reference entity.
  - The determinant attributes and their types in a transactional entity.
  - The determinant on the transactional entity that controls the setID of a shared reference entity.

- The setID-related database tables and views should be available as described in \textit{About Set Configuration Tables}.

- Ensure that setID seed metadata has been configured as follows:
  - The Reference Groups have been defined by teams and approved through the SetID Design Intent Repository process, then seeded in the standard reference groups table.
  - All set-enabled lookup types have been identified and approved through the SetID Design Intent Repository process, and then properly seeded in the standard lookup types table.

For more information, see \textit{Understanding SetID Machinery}.

\section*{8.2.1 How to Annotate Reference Entity Objects for Sharing}

\textbf{Note:}

These annotations are required only for \textit{set-enabled} non-lookup reference entities. Lookup references and set-enabled lookup references use a predefined lookups pattern; the reference group name is retrieved directly from database by the lookup code and the view application ID that are set on the foreign key reference of the transactional reference.

After building an entity object for a shared reference entity, you annotate the entity object.

To annotate the reference entity object, do the following:

1. Double-click the entity object to access its properties, as shown in \textit{Figure 8-2}.
2. In the Applications section of the Property Inspector, specify the name of the setID reference group to which the entity object belongs.

   For more information, see About Reference Groups.

3. Specify the setID pattern that the reference entity should use. There are three setID partitioning patterns. Choose one of these patterns based on your business requirements:
   
   • Row Striping (this is the default value)
   • Striping With Common Rows
   • Subscription

   When you select the SUBSCRIPTION pattern, the SetID Reference Table Pattern field appears. Specify the subscription table to use.

   **Caution:**

   The primary key columns of the setID subscription table must be named the same as those in the reference entity, and the setID column must be named SET_ID.

   For more information about these options, see About Partitioning Patterns.

4. In the entity object attributes, ensure that the entity object setID attribute that corresponds to the SET_ID database column is named SetId, as shown in Figure 8-3.
Caution:

If you cannot use the SET_ID column as your setID attribute, you must ensure that the attribute you use is named SetId, even if the database column is named differently. This applies to both entity objects and view objects.

8.2.2 Guidelines for Building Entity Associations for All Foreign References

After building an entity object for a transactional entity, you must create entity associations for all foreign references, including FND lookups. Because SET_ID must not be part of the primary key for any shared reference table (except FND_LOOKUP_VALUES), there is nothing unusual about associations for shared references.

Follow these guidelines when creating the associations:

- Make sure a destination accessor is generated for the association. ADF Business Components will not honor the association if you do not generate a destination accessor.
- Because these reference entities are non-composite, you should not generate source accessors.

8.2.3 How to Annotate Transactional Entity Objects for SetID

After you create entity associations for foreign references, you annotate the transactional entity object.

To annotate the transactional entity object, do the following:

1. Double-click the entity object to access its properties, as shown in Figure 8-4.
2. In the Applications section of the Property Inspector, designate which attributes are setID determinants for the table. For more information, see About SetID Determinant Types.

For every attribute that you want to use as a setID determinant, specify the corresponding determinant type.

To access setID determinant types programatically, use the following codes:

### Table 8-1  SetID Determinant Type Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Determinant Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>Asset Book</td>
</tr>
<tr>
<td>BU</td>
<td>Business Unit</td>
</tr>
<tr>
<td>CST</td>
<td>Cost Organization</td>
</tr>
<tr>
<td>PU</td>
<td>Project Unit</td>
</tr>
<tr>
<td>SET</td>
<td>Reference Data Set</td>
</tr>
</tbody>
</table>

**Note:**

- The attribute you need to use as a determinant might not exist on the parent record where its value can be retrieved from some context. In that case, you must create a transient attribute to represent the determinant, set the **SetId Determinant Type** property for it, then override the getter method of the transient attribute to get the value from wherever it has been stored.

- After a SetId Determinant code is defined, you can change the code in the EO.xml file.

3. For foreign keys that are setID enabled, specify the determinant attribute that drives each foreign key reference, as shown in Figure 8-5.

The default value of the setID determinant attribute is the default determinant type of the reference entity group targeted by the association that is defined for the foreign key.
Note:
If the determinant value is not directly available on the transaction table, you must create a transient attribute to model it, and ensure that the attribute is correctly populated.

Figure 8-5  SetID Determinant Attribute for a Foreign Key

Caution:
If the reference data has a composite key, you must specify the SetId Determinant Attribute property for the first attribute of the composite key.

8.2.4 About Defining View Accessors for Shared Reference Entities
Create a view accessor from the transaction entity to the reference entity.

8.2.5 How to Define a Key Exists Validator for Shared Reference Entities
The key exists validator will include the mapping of the foreign key attributes in the transactional entity to the corresponding attributes in the reference view accessor.
Caution:

If an attribute in your transactional entity was defined with null values allowed, the validator that you create will skip that attribute, and the end user will receive no indication of any problem. To ensure that the attribute is validated, you must edit the attribute and select the Mandatory checkbox in the attribute properties.

To define a keys exists validator, do the following:

1. Open the entity object for editing.
2. On the Attributes tab, select the foreign key attribute and add a validation rule. The Edit Validation Rule page appears, as shown in Figure 8-6.
3. At the top of the page, select a rule type of Key Exists.

Figure 8-6    Foreign Reference Validation Rule Definition

4. On the Rule Definition tab, select ViewAccessor as the validation target type.
5. Select the entity object lookup code attribute on the left-hand list, and the corresponding view accessor validation target lookup code attribute on the right-hand list.
6. Click Add to include the attribute pair on the mapping list.
7. Select the Validation Execution tab, as shown in Figure 8-7.
Because the validation should be executed every time the determinant value changes, it should be specified as a triggering attribute.

**Figure 8-7  Foreign Reference Validation Triggering Attributes**

![Edit Validation Rule for: WorkerAssignmentsID]

- **Validation Level**
  - The rule can be executed once at the transaction level for all entities, or at the entity level.
  - **Execute at Entity Level**
  - **Defer execution to Transaction Level**

- **Conditional Execution**
  - To execute this rule only when certain conditions are true, provide a boolean expression.
  - When no condition is provided, this rule always executes.

- **Triggering Attributes**
  - Execute only if one of the selected attributes has been changed.

<table>
<thead>
<tr>
<th>Available Attributes</th>
<th>Selected Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssignmentId</td>
<td></td>
</tr>
<tr>
<td>AssignmentName</td>
<td></td>
</tr>
<tr>
<td>BusinessUnit</td>
<td></td>
</tr>
<tr>
<td>EmploymentCategory</td>
<td></td>
</tr>
<tr>
<td>License</td>
<td></td>
</tr>
<tr>
<td>RequirementsRegion</td>
<td></td>
</tr>
</tbody>
</table>

---

**Note:**

The foreign key attributes that were mapped on the Rule Definition tab are by default added as triggering attributes.

8. In the Triggering Attributes section, select the determinant attribute from the left-hand list and shuttle it to the right-hand list.

9. Optionally, on the Failure Handling tab, specify a failure error message.

10. Click **OK** to create the key exists validator.

### 8.2.6 How to Create a Transient SetID Attribute

To create a transient setID attribute, do the following:

1. Create a new transient setID attribute to map to the SetId attribute on the reference entity, as shown in **Figure 8-8**.
2. Set the Type to Long.

3. Deselect the Persistent checkbox.

4. In the Updatable section, select Never, then click OK to create the transient attribute.

5. On the Java tab, generate or edit a Java class for the transaction entity object.

6. Because the setID value is computed at runtime based on the values of the reference group name, determinant type and determinant value, you must modify the transaction entity object's Java code to return the setID value at runtime.

Open the transactional EOImpl class and edit the getter method of your transient setID attribute to pass in the corresponding foreign key attribute name. For example:

```java
/** This method gets the attribute value for TransientSetIdAttr, using the alias name TransientSetIdAttr. */
public Long getTransientSetIdAttr() {
    return this.getSetId("SalaryCode");
    //"SalaryCode" is the foreign key attribute name on the WorkerAssignments transactional entity
}
```

In this example, you open the WorkerAssignmentsImpl.java class and edit getTransientSetIdAttr() to pass in the attribute name "SalaryCode" so its value will be returned at runtime.

**8.2.7 How to Create LOVs for Shared Reference Entities**

ADF Business Components supports defining LOVs at the attribute level in view objects.

To build a lookups LOV for a set-enabled reference entity:
1. Create an LOV on the foreign key attribute in the attribute wizard.

   The default LOV name is typically kept as \texttt{LOV\_attribute\_name}, as shown in Figure 8-9.

   \textbf{Figure 8-9} \hspace{1em} \textit{LOV Definition for a Set-Enabled Reference Entity}

2. Choose a view accessor from the list of available view accessors.

   Typically you will choose the same view accessor which was defined for the underlying entity object and used for the Key Exists validator.

3. Select an attribute from the view accessor to validate against, and ADF Business Components will automatically add that attribute to the list of return values.

4. Optionally, you can specify additional attributes to be returned to the master row when an LOV entry is selected.

5. Optionally, you can customize the LOVs UI hints by clicking \texttt{Edit List UI Hints} to access the List UI Hints dialog.

8.3 \textbf{About Integrating SetID Task Flows into Oracle Fusion Functional Setup Manager}

Every application registers task flows with a product called \textit{Oracle Fusion Functional Setup Manager}. Functional Setup Manager provides a single, unified user interface that enables implementers and administrators to configure all multiple applications by defining custom configuration templates or tasks based on their business needs.

The Functional Setup Manager UI enables customers and implementers to select the business processes or products that they want to implement. For example, an HR application can register setup activities like "Create Employees" and "Manage Employee Tree Structure" with Functional Setup Manager.
There is an application task flow for managing reference data sets, and one for managing reference data set assignments. To make these task flows available to application developers, implementers or administrators, you can register the appropriate task flow with Functional Setup Manager, using the parameters listed for each task flow in Table 8-2.

### Table 8-2  SetID Task Flows and Parameters

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Parameters Passed</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Reference Data Sets</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/setid/publicUi/flow/ManageSetIdSetsTF.xml#ManageSetIdSetsTF</td>
<td>To optionally specify a page heading for the task flow: pageTitle=&quot;titlestring&quot;</td>
<td>This task flow enables you to create and update reference data sets (setIDs and codes).</td>
</tr>
<tr>
<td>Manage Reference Data Set</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/setid/publicUi/flow/ManageSetIdAssignmentsTF.xml#ManageSetIdAssignmentsTF</td>
<td>To invoke the task flow: determinantType=type</td>
<td>This task flow enables you to manage reference data set assignments for a particular determinant type.</td>
</tr>
<tr>
<td>Assignments</td>
<td></td>
<td>To optionally restrict the page to assignments for a single reference group:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>referenceGroupName=name</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>To optionally specify a page heading for the task flow: pageTitle=&quot;titlestring&quot;</td>
<td></td>
</tr>
</tbody>
</table>

For information about implementing your specific product family, do the following:


2. See the Implementing Common Features guides for your product family.
This chapter describes the Oracle Fusion Middleware extensions for Oracle Applications base classes that extend the features of standard ADF Business Components classes.

The chapter includes the following sections:

- Introduction to Extensions for Oracle Applications Base Classes
- Understanding Multi-Language Support Features
- About Using WHO Column Features
- About Using PL/SQL-Based Entities
- About Accessing FND Services
- About Using Unique ID
- About Using Data Security
- About Using Document Sequencing

9.1 Introduction to Oracle Fusion Middleware Extensions for Oracle Applications Base Classes

Oracle Fusion Middleware extensions for Oracle Applications base classes provide additional features that are not part of the standard ADF Business Components core entity objects, view objects, and application modules.

The Oracle Fusion Middleware extensions support the following standard Oracle Applications features:

- TL (translatable) table
- WHO column
- PL/SQL entity
- FND services
- Unique ID
- Data security
- Document sequencing
The base classes extend ADF Business Components Entity, EntityDef, ViewObject, ViewRow, and ApplicationModule implementation classes.

The base classes provided by Oracle Fusion Middleware extensions are the following:

- OAApplicationModuleImpl
- OAEntityImpl
- OAEntityDefImpl
- OAVLObjectImpl
- OAViewRowImpl
- OAViewCriteriaAdapter

They are found in oracle.apps.fnd.applcore.oaext.model.package and extend the JBO classes with the same name (but without the OA prefix) in oracle.jbo.server.package.

In Oracle JDeveloper, selecting the Oracle Fusion Applications Developer role automatically sets the Oracle Fusion Middleware extensions for Oracle Applications base classes as the default classes for ADF Business Components objects. The base classes become available when you add the Applications Core library. For more information, see Set Up Your Development Environment.

9.2 Understanding Multi-Language Support Features

Multi-language support (MLS) gives Oracle the ability to ship its products in multiple languages by setting standards and guidelines for translation.

In JDeveloper, multi-language entities are those that maintain one or more translated attributes and require the storage of all relevant translations of these attributes. Such entities have a base table that has attributes that are not translated and do not vary by language (such as codes and IDs) and a TL table that has, in addition to the base table primary key, the translatable attributes for that entity (such as Display Name, and Application Name). Figure 9-1 illustrates this concept.

**Figure 9-1  Multi-Language Entity Tables**

For each row in the base table, there will be as many rows in the translation table as there are installed languages. The translation table's primary key is made up of the foreign key to the base table and a language column, which may be viewed as a foreign key to the FND_LANGUAGES table.

The translation table is fully populated. This means that rows for all installed languages are inserted even if the actual translations for these languages are not yet available. The logic, which maintains multi-language entities, is responsible for ensuring that the translation rows are inserted, updated, or deleted as required to meet the "fully populated" requirement. Translations, which have not been supplied, must be defaulted from one of the available translations. As updates occur to supply missing translations, the default values will be converted to true translations.

Since applications are run in a single language for any given user session, a convenient view is provided for the multi-language entities, which joins the base table and translation table and filters translations to the runtime language. This is the Multi-
language View. This view uses the userenv('LANG') expression to select the correct translation based on the session language, which usually comes from the NLS_LANG environment variable.

The following extensions support TL tables:

- OAEntityImpl
- OAViewRowImpl

As a developer, you can use multi-language extensions to deal with only one entity that contains both translatable and non-translatable attributes, instead of having to deal with two entities, one for the base table and one for the translation table.

Whenever an entity is created, the extensions ensure that the TL entities are also created for every installed language in the environment.

Whenever an insert is made into the base table or the table is updated, the same operations must also be performed on the corresponding TL table. Behind the scenes, the extensions override the appropriate ADF Business Components methods, such as create() and setAttribute(), to ensure that the TL table is populated correctly.

The extensions also enable you to work with only one ADF Business Components entity object at runtime for a multi-language database entity, and shield you from the two underlying tables (base and multi-language) that hold the data. You will see no inherent difference between a multi-language entity and a standard one. In addition, the extensions allow you to define an entity as multi-language in a JDeveloper design time environment, and provide any additional metadata for such that entity.

9.2.1 About Using Utility APIs

In addition, the following utility APIs are provided in OAEntityImpl:

- public boolean isTranslatable () - Returns true if this entity is a translatable entity.
- public boolean isTranslated () - Returns true if there is at least one translated language other than the base language for this entity.
- public String [] getTranslatedLanguages () - Returns an array of Language codes for which actual translations exist. The list always returns the base language as one of the translated languages. A record is considered translated if the LANGUAGE and SOURCE_LANG columns are equal.

The same set of APIs also will be provided on the OAViewRowImpl object, since it also would have the same characteristics of a row.

9.2.2 How to Create a Multi-Language ADF Business Components Entity Object

Creating a multi-language ADF Business Components entity object consists of four tasks:

- Creating an Entity Object for a _TL Table
- Creating an Entity Object for a Base Table
- Associating the _VL View and _TL Table Entity Objects
- Creating a View Object That Uses a Translatable Entity
9.2.2.1 Creating an Entity Object for a _TL Table

To create an entity object for translatable (_TL) tables, perform the following procedure.

Note:
This procedure does not apply to a _VL view. For information about creating an entity object for a _VL view, see Creating an Entity Object for a Base Table.

1. Name the entity `<Entity>TranslationEO`.
   For example, for a table named FND_ITEMS_DEMO_TL, you can name the entity `ItemsDemoTranslationEO`.

2. Include all of the table's attributes.
   Make sure the attribute for the LANGUAGE column is named `Language`, and the attribute for the SOURCE_LANG column is named `SourceLang`.
   If your TL table columns for LANGUAGE and SOURCE_LANG are named differently, it is important that you still name the attributes `Language` and `SourceLang`.

3. Identify the table's primary keys, including the LANGUAGE column.

4. Verify that this extends `OAEntityImpl` like any other entity object.

5. Add whatever validation logic you need for this entity and its attributes.
   The translatable values are unlikely to need any special validation.

Overriding the default attribute behavior:
By default, all the attributes in the _TL table will be considered translatable if they are:
- not a primary key attribute
- not an entity accessor
- one of the following types: VARCHAR, CHAR, FIXED_CHAR, LONGVARCHAR, CLOB

Note:
`SourceLang` and `Language` are special attributes and are handled by Oracle Fusion Middleware Extensions for Applications.

9.2.2.2 Creating an Entity Object for a Base Table

To create an entity object for a base table, perform the following procedure.

1. Name the entity object.
   Use the regular entity object naming convention. For example, for the FND_ITEMS_DEMO table, the corresponding entity would be named `ItemsDemoEO`. The entity should be based on the _VL view.
2. Include all columns except the RowId pseudo-column in the view.

3. Identify your primary keys as you normally would.

4. Set the entity-level Oracle Fusion Middleware Extensions for Applications schema-based ADF Business Components property named fnd:OA_BASE_TABLE with a value that names the true base table of your translatable entity.

For example, for the FND_ITEMS_DEMO_VL view, this value would be set to FND_ITEMS_DEMO_B.

You could use the entity Property Inspector to set this property, as shown in Figure 9-2.

**Figure 9-2  Entity Property Inspector**

Oracle Fusion Middleware Extensions for Applications automatically overrides the entity’s doDML() method to ensure that all inserts, updates, and deletes are actually performed on the base table identified by this property. All reads will be done against the _VL view.

9.2.2.3 Associate the _VL View and _TL Table Entity Objects

To create the association between the _VL view and _TL table entity objects, perform the following procedure.

1. Follow the standard association object naming convention that describes the entity relationships. For example, ItemsToTranslation.

2. In the Structure window, choose the entity object. In this case, it is ItemsToTranslation.
3. In the Overview window, choose the **Relationship** option.

4. Designate the association as a **Composition Association** with a 1:* cardinality, as shown in Figure 9-3.

**Figure 9-3 Composition Association**

When you select **Composition Association**, be sure to uncheck **Implement Cascade Delete** and **Cascade Update Key Attributes** if they are selected.

5. Select the base entity as the source and the _TL entity as the destination.

Since the Applications Core OAEntityImpl class overrides the `remove()` method on the EntityImpl class to handle Translation rows deletion, **Cascade Delete** is not required.

6. Configure **Source Accessor** and **Destination Accessor**, as shown in Figure 9-4.

**Note:**

Ensure that the **Source Accessor** has been created prior to performing Step 6.
9.2.2.4 Create a View Object That Uses a Translatable Entity

When creating the view objects that will access your translatable tables, keep in mind the following:

- Always use the base entity object created for the _VL view. For example, `ItemsDemoEO`.
- Do not use the _Translation entity object `ItemsDemoTranslationEO` directly. For the purpose of any code that needs to access your translatable entity, you should treat the base entity object as the only entity object. Coordination between the base and Translation entities is handled automatically and the Translation entity should remain "invisible". Otherwise, you can treat your base entity object like any other entity object.

For a _TL table with no corresponding _B table:

There may be a rare case where you have a _TL table and _VL view and no _B table, because all of the attributes are translatable. If this occurs, do the following:

1. Define the base entity on the database view _VL.
2. Set the Applications Core schema-based property `fnd:OA_BASE_TABLE` to be the _VL view name.
3. Override `doDML()` for the base entity to do nothing. This is going to be a virtual entity that does not have an underlying database table.
4. Create the translation entity object and the composite association between the base entity and the translation entity as you would in the regular scenario.
The Translation EO in this scenario alone must also include the non-translatable attributes because the base entity’s doDML() does nothing. If the translation entity does not include non-translatable attributes, you might get exceptions saying the attribute is not populated.

5. Mark all the non-translatable columns in the _TL entity, i.e., non-string fields and non-primary keys, as explicitly translatable by setting OA Translatable to true in the Applications section of the Property Inspector, as shown in Figure 9-5.

Figure 9-5   OA Translatable Setting

By default, only string fields (VARCHAR2 and its variants) are identified as translatable automatically by the parent. Primary key changes on the entity are also handled automatically by the framework. This means any numeric, date, or other data type attributes that are not primary key need to have the OA Translatable property set explicitly to true.

There is a slight downside to this approach as non-translatable columns (like numbers and dates), technically, are being marked as translatable. However, this approach is required in order to ensure attributes set on the base entity are propagated to the TL entity; otherwise, you will get an “attribute not populated” exception. This is needed because the base entity is virtual and the doDML() method on the base entity is empty.

9.2.2.5 What You Need to Know About Overrides

If you happen to override the create(AttributeList attributeList) method on your entity, do not forget to call super.create(attributeList) in the override method before invoking custom code. This is true in all scenarios.
9.3 About Using WHO Column Features

The WHO feature reports information about who created or updated rows in Oracle Applications tables. Oracle Applications upgrade technology relies on WHO information to detect and preserve customizations. ADF Business Components provides the ability to track the creation of an entity or the changes made to one.

The OAEntityImpl populates the WHO columns automatically. In addition to the standard history columns supported by ADF Business Components, the extension provides support for Last Update Login field.

All WHO columns are updated based on the current User Session. Table 9-1 lists the WHO columns and their descriptions.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Null?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATED_BY</td>
<td>VARCHAR2(64)</td>
<td>NOT NULL</td>
<td>Keeps track of which user created each row.</td>
</tr>
<tr>
<td>CREATION_DATE</td>
<td>DATE</td>
<td>NOT NULL</td>
<td>Stores the date on which each row was created.</td>
</tr>
<tr>
<td>LAST_UPDATED_BY</td>
<td>VARCHAR2(64)</td>
<td>NOT NULL</td>
<td>Keeps track of who last updated each row.</td>
</tr>
<tr>
<td>LAST_UPDATE_DATE</td>
<td>DATE</td>
<td>NOT NULL</td>
<td>Stores the date on which each row was last updated.</td>
</tr>
<tr>
<td>LAST_UPDATE_LOGIN</td>
<td>VARCHAR2(32)</td>
<td></td>
<td>Stores the Session ID of the user who last updated the row.</td>
</tr>
</tbody>
</table>

9.3.1 How to Use the Extension

In order for Oracle Fusion Middleware Extensions for Applications to populate your WHO columns automatically, do the following:

1. Ensure that your WHO column attributes are of the appropriate History Column type by using the Entity Attribute Wizard, as shown in Figure 9-6.
In the example entity, the WHO column **LastUpdateDate** is identified as a **modified on History Column** type.

2. Identify the following attributes as indicated:
   - **LastUpdatedBy** - modified by
   - **CreationDate** - created on
   - **CreatedBy** - created by

3. Ensure that the **LAST_UPDATE_DATE** and **CREATION_DATE** WHO columns have the Type as Timestamp (java.sql.Timestamp), as shown in Figure 9-7.
9.3.2 What Happens with WHO Column at Design Time and Runtime

WHO column features provide the following design time and runtime support:

- The extension supports the LAST_UPDATE_LOGIN column and ensures that the other columns are populated correctly.

- The LAST_UPDATED_BY and CREATED_BY columns are populated with a value based on the user name, and not with the user GUID, a user ID, or a session ID. To obtain the value to populate these columns in PL/SQL, use FND_GLOBAL.WHO_USER_NAME. In Java, the CreatedBy and LastUpdatedBy attributes will normally be populated automatically with the correct value by the base classes, or you can also obtain the value from OAEntityImpl.getWhoUser().

- History is provided for the Session ID of the user who last updated the row.

- Proper shaping in the Oracle Fusion Applications Developer role to make this history available.

9.4 About Using PL/SQL-Based Entities

PL/SQL entities are those that depend on PL/SQL packages to handle their Data Manipulation Language (DML) operations (insert, delete, update, and lock). Since Oracle Applications has a large amount of their business logic in PL/SQL and a lot of teams still use it, they need a mechanism that will allow them to use their PL/SQL code when building ADF Business Components entities. The Oracle Fusion Middleware extensions provide the following:

- A way to identify a PL/SQL entity using a custom property

- TL table support and the ability to override the appropriate DML operation
9.4.1 About Using APIs to Facilitate DML Operations

In addition, the following APIs are provided in the OAEntityImpl class to facilitate the insert, update, and delete DML operations in PL/SQL:

- protected void insertRow();
- protected void updateRow();
- protected void deleteRow();

The default implementations of these methods delegate to super.doDML(operation), which will result in SQL insert/update/delete being called for the entity.

9.4.2 How to Use the Extensions

A PL/SQL-based entity object provides an object representation of the data from a table or view and routes the DML operations to stored procedures in the database.

To identify an entity as a PL/SQL one, a custom attribute, OA_PLSQL_ENTITY, must be set to Y (Yes). This allows the framework to identify this entity as PL/SQL based. Do the following:

1. From the Applications window, choose an entity object.
2. In the Structure window, highlight the entity object.
3. From the Property Inspector tab, choose the Applications option.
4. Under PL/SQL, select Y from the OA PLSQL Entity dropdown menu, as shown in Figure 9-8.
5. Override the following methods for its DML operations and provide JDBC calls when applicable:
   - void insertRow();
   - void updateRow();
   - void deleteRow();

   Use the PL/SQL entity objects only if you have legacy PL/SQL code that maintains all your transactions and validations. If you are working on a new product and/or do not have a lot of PL/SQL legacy code, Applications Core recommends the use of Java entity objects over PL/SQL entity objects.

6. Call your PL/SQL insert, update, or delete procedure in your void insertRow();, void updateRow();, or void deleteRow(); method without calling super().

7. Create a callable statement to invoke your PL/SQL method.

8. Validate your attributes. You can do this in either of two places:
   - In your insertRow() or updateRow() methods: Perform your validation in Java in either of these two methods, or in PL/SQL stored procedures called from the methods.
   - In your validateEntity() method: If validations are done in a separate stored procedure in PL/SQL, you can call that stored procedure in this method.
9.4.3 What Happens with PL/SQL Entities at Design Time and Runtime

The extensions provide the following design time and runtime support:

- Provides the ability to identify PL/SQL-based entities.
- Invokes PL/SQL for DML operations.

9.5 About Accessing FND Services

Oracle Fusion Middleware extensions for Oracle Applications provide the following services:

- Profile
- Lookup
- Message
- Language
- Application
- Taxonomy
- Data Security
- Attachments

Oracle Fusion Middleware extensions provide an easy way to access these services and to invoke them. Typically, the services are provided as application modules. An application module serves as a container for the various view objects and provides business-service-specific functionality.

The services listed above are provided as a service-specific application module. For example, Profile functionality is made available in ProfileService.

Access to these services is provided as a getFNDNestedService (String service) method in the OAApplicationModuleImpl class. The OAApplicationModuleImpl extension is used to support access to the services.

See About Using the Extension, for implementation information.

9.5.1 About Using the Extension

The code in Example 9-1 shows how to provide access to an FND service. In this case, it is ProfileService.

OAConstants exposes the various service names as a constant.

Note that the getFNDNestedService () is just a utility method that looks up the rootAM and checks to see if an instance of the requested service already exists in the rootAM as a nested AM. If one exists, it will return it; if it does not, it will instantiate a new AM for that service that will be nested inside the rootAM and return it.

Example 9-1 Accessing an FND Service

ProfileService profileService = (ProfileService) myAM.getFNDNestedService (OAConstants.PROFILE_SERVICE);
// now call profile specific methods on the ProfileService AM
String appsServletAgent = profileService.getProfile("APPS_SERVLET_AGENT");
9.6 About Using Unique ID

In order to avoid primary key collision issues when synchronizing with disconnected clients, Oracle Applications standards require that an ADF Business Components entity object's primary key be populated with a Unique ID.

Oracle Fusion Middleware extensions support Unique ID by allowing an entity attribute to be populated with a globally unique value. The Fusion Unique ID Generator provided by ADF Business Components does this. The Unique ID can be used to populate an entity attribute of the BigDecimal and Long data types. The Unique IDs generated are of the BigDecimal type and meet certain criteria for uniqueness across database instances.

---

**Note:**

The database table column data type that corresponds to the entity attribute requiring a Unique ID must be large enough to hold the uniquely generated value. Typically, it should be of type NUMBER (18). NUMBER (15) may not be sufficient to hold the uniquely generated values.

In addition, Oracle Applications coding standards require that the entity attribute populated with a Unique ID be of type Long.

---

9.6.1 About Using the Extension

Oracle Fusion Middleware extensions provide both design time and runtime support for Unique ID.

9.6.2 What Happens with Unique ID at Design Time

At design time, Oracle Fusion Middleware extensions provide the ability to identify if an entity attribute needs a globally Unique ID. This is accomplished by setting Application Unique ID to true in the entity attribute's Property Inspector section, as shown in Figure 9-9.
9.6.3 What Happens with Unique ID at Runtime

Based on the design time setting, the framework populates the entity attribute with a globally unique value at runtime. This is accomplished by setting the following transient expression on the entity attribute's definition:

```java
oracle.jbo.server.uniqueid.UniqueIdHelper.getUniqueId(adf.object.unwrapObject());
```

9.7 About Using Data Security

Any custom view criteria adapter created by a product team will need to extend the `OAViewCriteriaAdapter` class in order for Data Security to work correctly.

By setting custom ADF Business Components properties at runtime, the `OAApplicationModuleImpl` class establishes the `OAViewCriteriaAdapter` class as the standard view criteria adapter for the ADF Business Components container.

9.7.1 How to Use the Extension

Product teams can do the following to create and use a custom view criteria adapter:

1. Extend `OAViewCriteriaAdapter` and invoke super methods for use cases not handled by the custom view criteria adapter.

2. Set the custom view criteria adapter by invoking the `setViewCriteriaAdapter()` method in the `create()` method of the custom `ViewObjectImpl` class.

3. Set the custom view criteria adapter on the `ViewObject`. 
9.8 About Using Document Sequencing

Document sequencing is a way to uniquely identify all business documents and business events belonging to a legal entity.

Document-sequence numbering has many country-specific requirements. It is a legal requirement in many EMEA, Asia Pacific, and Latin American countries. In the United States and the United Kingdom, it is used for internal control purposes and for financial-statement and other audits.

For more information about ADF Business Components integration of this feature provided by Oracle Fusion Middleware extensions, see Setting Up Document Sequences.
This chapter discusses how to use lookups for providing lists of values (LOVs) for application end users to select from, and for performing validation of newly entered data. It also discusses how to share lookup data across organizations by using setIDs to partition the data into different sets of LOVs. Each organization can then maintain its lookups in a common table, using LOVs specific to that organization.

This chapter includes the following sections:

- Introduction to Lookups
- About Preparing Entities and Views for Lookups
- About Referencing Lookups
- About Defining Validators for Lookups
- Annotating Lookup Code Reference Attributes for Set-Enabled Lookups
- About Integrating Lookups Task Flows into

10.1 Introduction to Lookups

Lookups in applications are used to represent a set of codes and their translated meanings. For example, a product team might store the values ‘Y’ and ‘N’ in a column in a table, but when displaying those values they would want to display "Yes" or "No" (or their translated equivalents) instead. Each set of related codes is identified as a lookup type. There are many different examples of these across multiple applications.

Lookups Implementation

When you have completed the development process as discussed in this chapter and have delivered your application with the ability to use lookups, application implementers and administrators must then be able to define and maintain lookups that are appropriate to the organization that will use the application. They can accomplish these tasks using the Manage Standard Lookups, Manage Set-Enabled Lookups and Manage Common Lookups applications.

You make these setup applications available to implementers and administrators by incorporating their task flows into Oracle Fusion Functional Setup Manager. For more information, see About Integrating Lookups Task Flows into.

10.1.1 Overview of Lookups

Lookups are codes that are defined in the global FND_LOOKUP_VALUES table, which is striped into multiple virtual tables using a VIEW_APPLICATION_ID column. Each of these virtual tables is thus identified as a view application. Each view application is exposed as a database view, and all have separate ADF Business Components. It is the responsibility of the team who owns a particular view
application to provide both the database view and the necessary ADF Business Components objects. Only these view definitions, and any validation code supporting them, should access the underlying lookups tables directly. All other code that references lookups should always go through the database views and their supporting ADF Business Components objects, never directly referencing either the lookups tables or their base classes.

Note:
If you have a custom view application, you need to prepare a custom lookup view. For more information, see How to Prepare Custom Lookup Views.

Lookup codes are identified in an application by the following keys:

- A lookup view, which defines a distinct set of lookup types.
  Each lookup view is accessed through its own view, and may have different attributes or different validation, almost as if it were a separate table.

- A lookup type, which is a string identifier of a type that groups certain codes together; for example, COLORS.
  Within each lookup type, multiple lookup codes can be defined. Example 10-1 shows sample code for defining multiple lookup codes.

- A lookup code, which is a string identifier for a code within a type; for example, RED.

- A set or setID (for set-enabled lookups), which identifies the reference data set to which the lookup code belongs.
  For more information about setIDs, see Managing Reference Data with SetIDs.

The FND_LOOKUP_TYPES table defines the lookup types available.

Note:
When you register a lookup view application, you set a SET_ENABLED flag to indicate that the lookup view is set enabled. For this to be valid, every lookup type within that lookup view must have a reference group defined. The reference group is part of the lookup definition, and was defined when the lookup was defined. How that happens is beyond the scope of this documentation.

A reference to a non set-enabled lookup can be implemented exactly like any other foreign key reference, by specifying the lookup type in the view criteria. For set enabled lookups, you must specify the following additional properties, but only to add the indirection through the setID metadata:

- Indicate the view application ID and lookup type for lookup code attributes.
- Indicate the determinant attribute and determinant type, if the lookup type is set-enabled.

The use of setID metadata allows for the use of generic lookup entity objects, because the lookup type is automatically bound based on the metadata that you provide.
### Example 10-1   Defining Multiple Lookup Codes

View Application = 0 (FND_LOOKUPS)
- Lookup Type = COLORS "Colors"
  - Lookup Code = RED "Red"
  - Lookup Code = YELLOW "Yellow"
  - Lookup Code = GREEN "Green"

View Application = 3 (FND_COMMON_LOOKUPS)
- Lookup Type = COLORS "Colors"
  - Lookup Code = MAGENTA "Magenta"
  - Lookup Code = CHARTREUSE "Chartreuse"
  - Lookup Code = AQUAMARINE "Aquamarine"

### 10.1.2 About Standard, Set-Enabled, and Common Lookup Views

All lookups business objects exist in the `publicEntity` subpackage of the `oracle.apps.fnd.appcore.lookups.model` package. They can be imported into any Oracle JDeveloper application through `Lookups-Model.jar`. They are as follows:

**Lookup Types**

- **Entity Object:** `LookupTypePEO`
- **View Object:** `LookupTypePVO`
- **Base Table/View:** `FND_LOOKUP_TYPES_VL`

Each lookup type defines a set of lookup codes, and describes the intended usage of that set of codes. Note the `FND_LOOKUP_TYPES_VL` table and ADF Business Components objects are only meaningful when a `VIEW_APPLICATION_ID` is specified to choose the view application. You should never use either the table or the view without supplying the `VIEW_APPLICATION_ID`.

Product teams that own a view application must expose a pre-defined view for lookup types, exposing only the lookup types appropriate to their view application.

---

**Note:**

Product teams that own a view application also are responsible for providing the service, the loader, the UI, and the database view.

---

If your product has no special validation requirements, you can place your lookups in one of the central lookup views such as `FND_LOOKUPS`. However, if you define your own view application, you must supply a database view to match it.

**Lookup Values**

- **Entity Object:** `LookupValuePEO`
- **View Object:** none
- **Base Table/View:** `FND_LOOKUP_VALUES_VL`

The `FND_LOOKUP_VALUES_B` table (along with `FND_LOOKUP_VALUES_TL`) is the primary table that stores all the different lookup codes.

The `FND_LOOKUP_VALUES_VL` view is extended by the views in the three following listings (`FND_LOOKUPS`, `FND_COMMON_LOOKUPS`, and `FND_SETID_LOOKUPS`). If you want to define your own product specific lookups, you should extend this view as
well. This object contains the subset of columns that are expected to be common to all views that extend from this, with any additional columns required being added on an as-needed basis.

These objects should only be referenced by lookup view application owners when defining their own views and ADF Business Components objects. All other references should go through the objects created for that lookup view. The three standard ones that Oracle ships are FND lookups, common lookups, and setID lookups. If other products have lookup views, you should use the entity objects and view objects provided for them by the owning team.

(FND) Lookups

- **Entity Object**: LookupPEO
- **View Object**: LookupPVO
- **Base Table/View**: FND_LOOKUPS

The naming of the lookup objects can get confusing; the Lookups object is intended to refer specifically to FND lookups. The Lookup Values object in the previous listing is the generic object. The FND_LOOKUPS view is primarily used to store FND-specific lookup values but is also used to store lookup values that are common across multiple applications. For example, the "Yes/No" example given in the overview might be used by multiple product teams, so to avoid duplication that code can be stored centrally in FND_LOOKUPS.

This view extends from the FND_LOOKUP_VALUES_VL view, but only selects rows that have VIEW_APPLICATION_ID = 0 and SET_ID = 0.

Common Lookups

- **Entity Object**: CommonLookupPEO
- **View Object**: CommonLookupPVO
- **Base Table/View**: FND_COMMON_LOOKUPS

**Note:**

This view also was used to store lookup codes that were common to multiple applications, but it now exists only for backward compatibility.

This view extends from the FND_LOOKUP_VALUES_VL view, but only selects rows that have VIEW_APPLICATION_ID = 3 and SET_ID = 0.

SetID Lookups

- **Entity Object**: SetIdLookupPEO
- **View Object**: SetIdLookupPVO
- **Base Table/View**: FND_SETID_LOOKUPS

This view is used to store lookup codes that are set-enabled. The meanings corresponding to the given lookup code will vary depending on the value of the setID determinant.

This view extends from the FND_LOOKUP_VALUES_VL view, but only selects rows that have VIEW_APPLICATION_ID = 2.
10.1.3 About Lookup Customization Levels

Customization levels are defined on lookup types and can be used to enforce pre-defined data security policies that restrict how and by whom lookup types and their codes can be edited.

Valid values for CUSTOMIZATION_LEVEL are defined in the standard lookup type 'CUSTOMIZATION_LEVEL'. Table 10-1 lists these values.

Table 10-1    CUSTOMIZATION_LEVEL Lookup Codes

<table>
<thead>
<tr>
<th>Lookup Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>User</td>
</tr>
<tr>
<td>E</td>
<td>Extensible</td>
</tr>
<tr>
<td>S</td>
<td>System</td>
</tr>
</tbody>
</table>

10.1.3.1 What Happens to Customization Levels at Runtime

At runtime, the customization levels are interpreted as follows:

**User**
- Insertion of new codes is allowed
- Updating of start date, end date, and enabled fields is allowed
- Deletion of codes is allowed
- Updating of tag is allowed

**Extensible**
- Deletion of lookup type is **not** allowed
- Insertion of new codes is allowed
- Updating of start date, end date, enabled fields, and tag is allowed **only** if the code is not 'seed data'
- Deletion of codes is allowed **only** if the code is not 'seed data'
- Updating of module is **not** allowed

**System**
- Deletion of lookup type is **not** allowed
- Insertion of new codes is **not** allowed
- Updating of start date, end date, and enabled fields is **not** allowed
- Deletion of codes is **not** allowed
- Updating of tag is **not** allowed
- Updating of module is **not** allowed
In each of these scenarios, 'seed data' means LAST_UPDATED_BY = 'SEED_DATA_FROM_APPLICATION'. Also, to allow seed data to be edited, these rules are not enforced if the current user is 'SEED_DATA_FROM_APPLICATION'.

10.2 About Preparing Entities and Views for Lookups

It is expected that the owner of a lookup view will produce entity objects and view objects based on the entity objects for standard lookups database objects; for example, HR_LOOKUPS, GL_LOOKUPS, OE_LOOKUPS and so on. These view objects will typically be used for lookup validation as well as LOVs. If you put your lookups in the standard lookup views, you do not have to define anything, but simply reference the objects that are already provided.

Additionally, multiple ViewCriteria may be exposed on the lookups view object to take care of date ranging the lookup by supplying bind parameters for start and end active dates.

For a description of lookups tables and views provided by Oracle Fusion Middleware Extensions for Applications and their corresponding public business objects, see About Standard_Set-Enabled_ and Common Lookup Views.

10.2.1 How to Prepare Custom Lookup Views

If you have a simple lookup with no special requirements, you are free to define it in the centrally provided lookup views. You do not have to create your own lookup view just because you have lookups. However, if you have special validation requirements that are not satisfied by the central lookup views, you might want to create a private lookup view. If you do choose to create your own lookup view, you must take responsibility for the additional work required to support your lookup view as described in the following sections.

In preparing lookup views, you must perform several decision-based tasks.

To prepare lookup views, do the following:

1. Decide whether you really need a private lookup view.

   If you have no need for special attributes, special validation, or a private namespace for lookup types, you can use one of the centrally defined lookup views (FND_LOOKUPS, FND_COMMON_LOOKUPS, and FND_SETID_LOOKUPS). All of these lookup views are available for any product to use. If none of the central views meet your needs, you may define your own.

   **Note:**
   
   If you are using any of the three central lookup types (FND_LOOKUPS, FND_COMMON_LOOKUPS, and FND_SETID_LOOKUPS), you can skip the rest of this section.

   Lookup views are owned by applications (as determined by the view_application_id). There can be only one lookup view per view_application_id. It is up to the owner of the lookup view to make the view available for other applications to use, or to designate the lookup view as private.

2. Decide whether your lookup view should be set enabled.
If so, you must expose set_id as part of the "primary key" of your lookup view, and all references to it will have to include, either directly or indirectly, the set_id to use.

3. Define a database view to expose the lookup types included in your lookup view.

At a minimum your view must select from the base FND_LOOKUP_TYPES_VL view, expose the internal name and the display name, and include "where VIEW_APPLICATION_ID = my_application_id" in the where clause. In addition, if your view is set enabled, the lookup types view must include the REFERENCE_GROUP_NAME column. You are free to join additional tables, add additional attributes, or add additional filters to the where clause as desired. A template for the view might be:

```sql
select LOOKUP_TYPE,
       MEANING, DISPLAY_NAME,
       REFERENCE_GROUP_NAME, /* Only if set enabled */
... from FND_LOOKUP_TYPES_VL
where VIEW_APPLICATION_ID = my_application_id
and ...
```

4. Define a database view to expose the lookup codes included in your lookup view.

At a minimum your view must select from the base FND_LOOKUP_VALUES_VL view, expose the lookup type, the lookup code internal name, and the lookup code display name, and include "where VIEW_APPLICATION_ID = my_application_id" in the where clause. In addition, if your view is set enabled, the lookup values view must include the SET_ID column as part of the primary key. You are free to join additional tables, add additional attributes, or add additional filters to the where clause as desired. A template for the view might be:

```sql
select LOOKUP_TYPE,
       LOOKUP_CODE,
       SET_ID, /* Only if set enabled */
       MEANING,
... from FND_LOOKUP_VALUES_VL
where VIEW_APPLICATION_ID = my_application_id
and SET_ID = 0 /* Only if not set enabled */
and ...
```

5. Register your lookup view application and database views.

All view applications and the views used to reference them must be registered in the FND_LOOKUP_VIEWS metadata table. To register your lookup views, write a SQL script that calls the FND_LOOKUPS_UTIL.REGISTER_LOOKUP_VIEWS PL/SQL API. For example:

```sql
begin
  fnd_lookups_util.register_lookup_views(
    p_view_application_short_name => 'FND',
    p_set_enabled => 'N',
    p_lookup_type_view => 'FND_STANDARD_LOOKUP_TYPES',
    p_lookup_code_view => 'FND_LOOKUPS');
end;
```

This script registers required seed data, and must be run on every database instance.
6. Create ADF Business Components objects for your lookup view.

Each lookup view should have a separate entity object and view object (or PEO and PVO) for both lookup types and lookup codes, extending from the base entity object and view object provided for FND_LOOKUP_TYPES and FND_LOOKUP_VALUES.

For more information, see About Standard Set-Enabled and Common Lookup Views.

10.3 About Referencing Lookups

You must create view accessors for all lookups data sources (FND_COMMON_LOOKUPS, FND_SETID_LOOKUPS, HR_LOOKUPS, and so on) that are referenced in the entity object.

10.3.1 How to Reference Lookups

Lookups that are referenced in the entity object must have view accessors.

To reference lookups, do the following:

1. Import the lookups standard view objects into your project and make sure they can be referenced.

2. Open the entity object for editing.

3. On the View Accessors tab, add a view accessor. The View Accessors page appears.

4. Select a view object from the left-hand list and shuttle it to the right-hand list, then specify an accessor name.

5. Select the new view accessor and click Edit. The Edit View Accessor page appears.

6. Select the view criteria to use (if available), specify an order-by, and provide the bind parameter value.

---

**Note:**

All set-enabled view accessors are row sensitive (the determinant on the master or transactional row affects the query); therefore the **Row-level bind values exist** check box must always be selected for set-enabled view accessors. For example, view accessors to FND_SETID_LOOKUPS (set-enabled lookups cases) must have **Row-level bind values exist** selected because the setID value may change row by row and affect the validation result. Hence, the ViewAccessor Row Set will need to be refreshed row by row.

7. Click **OK** twice to finish creating the view accessor.

10.4 About Defining Validators for Lookups

You must create a validator for every foreign reference in an entity object. For set-enabled reference entities, the validator must be created at the entity object level, not at the attribute level, because it has dependencies on other attribute values such as the setID determinant attribute.

The type of validator to use depends on the expected size of the rowset for a given lookup type:
For a lookup definition where the rowset returned for a lookup type or lookup code is expected to be less than approximately 100 rows, use a list validator. See How to Define a List Validator.

For a lookup definition where the rowset returned for a lookup type or lookup code is expected to significantly exceed 100 rows, use a key exists validator. See About Defining a Key Exists Validator.

**Caution:**

If an attribute in your transactional entity was defined with null values allowed, the validator that you create will skip that attribute, and the end user will receive no indication of any problem. To ensure that the attribute is validated, you must edit the attribute and select the Mandatory checkbox in the attribute properties.

### 10.4.1 How to Define a List Validator

You define a list validator for lookup definitions where the rowset returned for a lookup type or lookup code is expected to be less than 100 rows.

To define a list validator, do the following:

1. Open the entity object for editing.

2. On the Validators tab, add a validation rule for the entity. The Edit Validation Rule page appears, as shown in Figure 10-1.

3. At the top of the page, select a Rule Type of List.
4. On the Rule Definition tab, do the following:

   a. Select the foreign reference column that is the lookup code (for example, SalaryCode) as the attribute.

   b. Select In as the operator.

   c. Select View Accessor Attribute as the list type.

5. From the Select View Accessor Attribute list, select LookupCode as the view accessor validation target lookup code attribute.

6. Select the Validation Execution tab, as shown in Figure 10-2.

   Because the validation should be executed every time the determinant value changes, it should be specified as a triggering attribute.
Figure 10-2  Lookups List Validator Execution

Note:
The foreign key attributes that were mapped on the Rule Definition tab are by default added as triggering attributes.

7. In the Triggering Attributes section, select the determinant attribute from the left-hand list and shuttle it to the right-hand list.

8. Optionally, on the Failure Handling tab, specify a failure error message.

9. Click OK to create the list validator for this lookup.

10.4.2 About Defining a Key Exists Validator

The key exists validator will include the mapping of the foreign key attributes in the transactional entity to the corresponding attributes in the reference view accessor. There must be a foreign key attribute on the transactional entity for each primary key attribute on the reference entity. First, you must provide missing foreign key attributes in the form of transient attributes. Next, you can create the validator that uses those attributes.

10.4.2.1 How to Define a Transient Lookup Type

Do the following:

1. Open the transactional entity object for editing.
2. Create a new transient lookup type attribute to map to the LookupType attribute on the reference entity, as shown in Figure 10-3.

**Figure 10-3  New Transient Lookup Type Entity Attribute**

3. Set the **Type** to **String**.

4. Set the **Value Type** to **Expression**, and provide a constant value for the attribute.

5. Deselect the **Persistent** checkbox.

6. In the **Updatable** section, select **Never**, then click **OK** to create the transient attribute.

**10.4.2.2 How to Define a Key Exists Validator**

Do the following:

1. On the Validators tab, add a validation rule for the transactional entity. The Edit Validation Rule page appears, as shown in Figure 10-4.

2. At the top of the page, select a Rule Type of **Key Exists**.
3. On the Rule Definition tab, select a **Validation Target Type** of View Accessor.

4. Select the entity object lookup code attribute on the left-hand list, and the corresponding view accessor validation target lookup code attribute on the right-hand list.

   Click **Add** to include the attribute pair on the mapping list.

5. Select the entity object transient lookup type attribute on the left-hand list, and the corresponding view accessor validation target lookup type attribute on the right-hand list.

   Click **Add** to include the attribute pair on the mapping list.

6. Select the entity object transient setID attribute on the left-hand list, and the corresponding view accessor validation target setID attribute on the right-hand list.

   Click **Add** to include the attribute pair on the mapping list.

7. Select the Validation Execution tab, as shown in Figure 10-5.

   Because the validation should be executed every time the determinant value changes, it should be specified as a triggering attribute.
8. In the Triggering Attributes section, select the determinant attribute from the left-hand list and shuttle it to the right-hand list.

9. Optionally, on the Failure Handling tab, specify a Failure Message.

10. Click OK to generate the key exists validator for this lookup. Then, save your project.

10.5 Annotate Lookup Code Reference Attributes for Set-Enabled Lookups

These properties are used only for set-enabled lookups, and only to do setID indirection. The setID lookup type LOV will show only those lookup types that are defined in your specified view application ID.

Do the following to annotate lookup code reference attributes:

1. Edit your set-enabled transaction table entity object.

2. Annotate each lookup code reference with setID machinery metadata, as shown in Figure 10-6.
To specify an attribute for use as a lookup code reference, select the attribute in the entity object editor. On the Applications tab of the Property Inspector.

- For a set-enabled lookup type, specify which determinant attribute on the entity object drives the setID of this lookup reference.

- For a set-enabled foreign key, you should also specify the setID determinant attribute that drives the foreign key reference.

3. Specify the **SetID View Application Id** and **SetId LookUp Type** properties.

### 10.6 About Integrating Lookups Task Flows into Oracle Fusion Functional Setup Manager

Every Oracle application registers task flows with a product called *Oracle Fusion Functional Setup Manager*. Functional Setup Manager provides a single, unified user interface that enables implementers and administrators to configure all multiple applications by defining custom configuration templates or tasks based on their business needs.

The Functional Setup Manager UI enables customers and implementers to select the business processes or products that they want to implement. For example, an HR application can register setup activities like "Create Employees" and "Manage Employee Tree Structure" with Functional Setup Manager.

There are application task flows for managing common lookups, set-enabled lookups, and standard lookups. To make these task flows available to application developers,
implementers or administrators, you can register the appropriate task flow with Functional Setup Manager, using the parameters listed for each task flow in Table 10-2. These taskflows can be used to manage lookups in the centrally defined lookup views (FND_LOOKUPS, FND_COMMON_LOOKUPS, and FND_SETID_LOOKUPS). All other lookup views (and any associated taskflows) are owned by applications (as determined by the VIEW_APPLICATION_ID). Contact the owning application for instructions on managing lookups in their lookup views.

Table 10-2  Lookups Task Flows and Parameters

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Parameters Passed</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Standard Lookups</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/lookups/publicUi/flow/ManageStandardLookupsTF.xml#ManageStandardLookupsTF</td>
<td>To invoke search mode to query and edit lookup types and their codes in the Standard Lookups view: mode='search' To restrict search mode to Standard lookups belonging to a particular product module: mode='search' moduleType='moduletype' moduleKey='modulekey' To invoke edit mode for a single lookup type and its lookup codes: mode='edit' lookupType='lookuptype' To optionally specify a page heading for the task flow: pageTitle='titlestring'</td>
<td>This task flow enables you to create and edit lookups in the centrally owned Standard view (view application = 0).</td>
</tr>
<tr>
<td>Task Flow Name</td>
<td>Task Flow XML</td>
<td>Parameters Passed</td>
<td>Behavior</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Manage Set-Enabled Lookups</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/lookups/publicUi/flow/ManageSetEnabledLookupsTF.xml#ManageSetEnabledLookupsTF</td>
<td>To invoke search mode to query and edit lookup types and their codes in the Set Enabled Lookups view: mode='search'</td>
<td>This task flow enables you to create and edit lookups in the centrally owned Set Enabled view (view application = 2).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To restrict search mode to Set Enabled lookups belonging to a particular product module:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode='search' moduleType='moduletype' moduleKey='modulekey'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>To invoke edit mode for a single lookup type and its lookup codes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode='edit' lookupType='lookuptype'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>To optionally specify a page heading for the task flow:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>pageTitle='titlestring'</td>
<td></td>
</tr>
<tr>
<td>Manage Common Lookups</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/lookups/publicUi/flow/ManageCommonLookupsTF.xml#ManageCommonLookupsTF</td>
<td>To invoke search mode to query and edit lookup types and their codes in the Common Lookups view: mode='search'</td>
<td>This task flow enables you to create and edit lookups in the centrally owned Common Lookups view (view application = 3).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>To restrict search mode to Common lookups belonging to a particular product module:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode='search' moduleType='moduletype' moduleKey='modulekey'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>To invoke edit mode for a single lookup type and its lookup codes:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode='edit' lookupType='lookuptype'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>To optionally specify a page heading for the task flow:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>pageTitle='titlestring'</td>
<td></td>
</tr>
</tbody>
</table>
For information about implementing your specific product family, do the following:

1. Access the Oracle Fusion Applications library.

2. See the Implementing Common Features guides for your product family.
This chapter describes how to set up document sequences, which uniquely number documents, provide proof of completeness, and create audit trails.

This chapter contains the following sections:

- **Introduction to Document Sequences**
- **About Defining Document Sequence Categories**
- **About Assigning a Document Sequence**
- **About Striping Document Sequence Assignments**
- **About Defining a Document Sequence Audit Table**
- **About Enabling Document Sequences in ADF Business Components**
- **About Managing PL/SQL APIs**
- **About Integrating Document Sequence Task Flows into**

### 11.1 Introduction to Document Sequences

A document sequence uniquely numbers documents generated by an Oracle Fusion application. Using Oracle Fusion applications, you initiate a transaction by entering data through a form and generating a document, for example, an invoice. A document sequence generates an audit trail that identifies the application that created the transaction, for example, Oracle Receivables, and the original document that was generated, for example, invoice number 1234.

Document sequences can provide proof of completeness. For example, document sequences can be used to account for every transaction, even transactions that fail. Document sequences generate audit data, so even if documents are deleted, their audit records remain.

Document sequences can also provide an audit trail. For example, a document sequence can provide an audit trail from the general ledger into the subsidiary ledger, and to the document that originally affected the account balance.

There are three types of document sequence numbering:

- **Automatic** - Assigns a unique number to each document as it is generated. Automatic numbering is sequential by date and time of creation.

- **Gapless** - Automatically generates a unique number for each document, but ensures that the document was successfully generated before assigning the number. With gapless numbering, no sequence numbers are lost due to incomplete or failed document creation.
For gapless numbering, you also have the option of validating the transaction date. Validating the transaction date ensures that the sequence numbers and transaction dates are "chronologically correlated", that is, lower sequence numbers have been issued to earlier transaction dates. You enable this option on the Manage Document Sequence user interface.

**Note:**
It is recommend that you choose this type only when gapless numbering is essential, as it may affect the performance of your system.

- **Manual** - Requires a user to assign a unique number to each document before it is generated. With manual numbering, numerical ordering and completeness is not enforced. Users can skip or omit numbers when entering the sequence value.

Table 11-1 defines document-sequence terminology.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document Sequences</td>
<td>Document sequences are owned by a product and can be assigned to categories that belong to the same product as the sequence. Sequences can be automatic, manual, or gapless, and are effective within a date range.</td>
</tr>
<tr>
<td>Document Sequence Categories</td>
<td>A document sequence category belongs to a table, which is owned by a product. Document sequence categories are entered with either the System Administrator form (Payables and Cash Management) or product forms (General Ledger and Receivables).</td>
</tr>
<tr>
<td>Sequence Assignments</td>
<td>The user assigns a sequence for each category. The assignments are owned by determinant type and determinant value and are effective within a date range.</td>
</tr>
</tbody>
</table>

11.2 About Defining Document Sequence Categories

Document sequence categories organize documents into logical groups.

- A document sequence category is one of the rules you use to define which documents a sequence assigns numbers to.

- You can separately number each document sequence category by assigning a different sequence to each category.

A document sequence category identifies the database table that stores documents resulting from transactions your users enter. When you assign a sequence to a category, the sequence numbers the documents that are stored in a particular table.

11.3 About Assigning a Document Sequence

Before you can assign a sequence to number documents, you must define which documents are to be numbered.

Defining a sequence is different from assigning a sequence to a series of documents.

- A sequence's definition determines whether a document's number is automatically generated or manually entered by the user.
• A sequence’s assignment, that is, the documents a sequence is assigned to, is defined in the Sequence Assignments form.

11.4 About Striping Document Sequence Assignments

Document sequences can optionally be given a determinant type when defined. This determinant type specifies the context "dimension" that will be used to stripe assignments of this document sequence. If a determinant type has been specified, then each assignment of this document sequence must include a particular value (or "determinant value") for this context. At runtime, only those document sequence assignments with a determinant value matching the current value for this dimension in the user's session context will be visible. If no determinant type is defined, then all assignments are considered global and are always visible.

Table 11-2 shows the possible values for the determinant type.

Table 11-2 Determinant Type Values

<table>
<thead>
<tr>
<th>Determinant Type Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEDGER</td>
<td>Ledger</td>
</tr>
<tr>
<td>LE</td>
<td>Legal Entity</td>
</tr>
<tr>
<td>BU</td>
<td>Business Unit</td>
</tr>
</tbody>
</table>

For example, if a document sequence is defined with determinant type of "Ledger", then each assignment of this document sequence must include a particular Ledger value. At runtime, only assignments that match the current Ledger value will be visible.

11.5 About Defining a Document Sequence Audit Table

Each time a C or PL/SQL call is made to request the next document sequence value, this audit data is inserted into the corresponding product team’s document sequence audit table.

Product teams using FND Document Sequence need to create an audit table with a name whose format is `application_short_name_DOC_SEQUENCE_AUDIT`, where `application_short_name` is the name of the application. For example, "AR_DOC_SEQUENCE_AUDIT."

The audit table must contain the columns and types shown in Table 11-3.

Table 11-3 Audit Table Columns and Types

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOC_SEQUENCE_ID</td>
<td>NOT NULL</td>
<td>NUMBER(18)</td>
</tr>
<tr>
<td>DOC_SEQUENCE_VALUE</td>
<td>NOT NULL</td>
<td>NUMBER(15)</td>
</tr>
<tr>
<td>DOC_SEQUENCE_ASSIGNMENT_ID</td>
<td>NOT NULL</td>
<td>NUMBER(18)</td>
</tr>
<tr>
<td>CREATION_DATE</td>
<td>NOT NULL</td>
<td>TIMESTAMP(6)</td>
</tr>
<tr>
<td>CREATED_BY</td>
<td>NOT NULL</td>
<td>VARCHAR2(64 CHAR)</td>
</tr>
</tbody>
</table>
**Table 11-3 (Cont.) Audit Table Columns and Types**

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAST_UPDATE_DATE</td>
<td>NOT NULL</td>
<td>TIMESTAMP(6)</td>
</tr>
<tr>
<td>LAST_UPDATED_BY</td>
<td>NOT NULL</td>
<td>VARCHAR2(64 CHAR)</td>
</tr>
<tr>
<td>LAST_UPDATE_LOGIN</td>
<td></td>
<td>VARCHAR2(32 CHAR)</td>
</tr>
<tr>
<td>ENTERPRISE_ID</td>
<td>NOT NULL</td>
<td>NUMBER(18)</td>
</tr>
</tbody>
</table>

**Note:**

Document sequence auditing is optional. If the AUDIT_TABLE_NAME attribute is null (that is, if the "Audit" checkbox has been unselected in the user interface), audit data will not be inserted at runtime.

---

**11.6 About Enabling Document Sequences in ADF Business Components**

This section focuses on ADF Business Components integration of document sequences provided by Fusion Middleware extensions for Oracle Applications base classes.

**11.6.1 About Using the Document-Sequence Extension**

The document sequence is generated and validated in the `postChanges()` method of the `OAEntityImpl` class. In automatic mode, it is generated by calling the public API `Long getDocSequence(Long appId, String categoryCode, Long sobId, String methodCode, TimestamptxnDate, Long seqVal, String suppressWarn, String suppressError)` in the `OAEntityImpl` class. In manual mode, it is validated by calling `public void validateDocSequence(Long appId, String categoryCode, Long sobId, String methodCode, Timestamp txnDate, Long seqVal, String suppressWarn, String suppressError)` in the same class.

You do not need to do anything to get the default behavior of generation and validation of a document sequence. However, if you require some special behavior, such as additional validation or adding an additional prefix or suffix, you can override these methods.

The Javadoc for the key methods in `OAEntityImpl` is shown in **Example 11-1**.

**Example 11-1 Javadoc for OAEntityImpl**

```java
/**
 * Override of EntityImpl.postChanges() to handle document sequencing.
 * If an entity attribute has been identified that it should be populated
 * using a document sequence (in the Applications Property Inspector panel),
 * then at this point in the entity life cycle, we will populate the attribute
 * with a document sequence based on the inputs, provided the sequence method
 * is automatic. If the document sequence is manual, we will validate the
 * document sequence.
 * See parent class for complete documentation
 * @param e this Entity Object's transaction event.
 * @see #validateDocSequence
 * @see #getDocSequence
 * @see EntityImpl#postChanges
 */
```
public void postChanges(TransactionEvent e){...}

/**
 * Will populate the entity attribute with a document sequence in Automatic mode, based on the schema based properties being set on the attribute using Applications Property Inspector in the Entity Attribute editor in JDev for
 * fnd:DOC_SEQUENCE (Document Sequence),
 * fnd:DOC_SEQ_APPLICATION_ID (Application Id),
 * fnd:DOC_SEQ_METHOD_CODE (Method Code),
 * fnd:DOC_SEQ_CATEGORY_CODE (Category Code),
 * fnd:DOC_SEQ_DETERMINANT_VALUE (Ledger Id),
 * fnd:DOC_SEQ_DETERMINANT_TYPE
 * fnd:DOC_SEQ_TXN_DATE (Transaction Date)
 * Application Id, Method Code, Category Code, Ledger Id and Transaction Code,
 * should be populated with valid Groovy expressions.
 * The Groovy expressions when evaluated should return a Long for Application Id,
 * and Ledger Id, String for Method Code and Category Code, Timestamp for Transaction Date fields.
 * This method will be invoked by postChanges() method on the entity, when posting the data to the database.
 * Override this if you want a different behavior/way of populating the document sequence.
 * @param appId Application Id
 * @param categoryCode Document Sequence Category Code
 * @param sobId Ledger Id to use for this Document Sequence.
 * @param methodCode Document sequence Method Code (Automatic ("A"), Manual ("M") or null for both modes).
 * @param txndateDocument Transaction Date
 * @param seqVal Document Sequence Value to use in Manual Mode
 * @param suppressWarn Suppress warning (Y/N/null)
 * @param suppressError Suppress Error (valid values are Y/N/null)
 * @return Document Sequence Value
 * @see #postChanges
 */

public Long getDocSequence(Long appId, String categoryCode, Long sobId, String methodCode, Timestamp txnDate, Long seqVal, String suppressWarn, String suppressError)
{
...

/**
 * Will validate the entity attribute with a document sequence value to use in Manual mode, based on the schema based properties being set on the attribute using Applications Property Inspector in the Entity Attribute editor in JDev for
 * fnd:DOC_SEQUENCE (Document Sequence),
 * fnd:DOC_SEQ_APPLICATION_ID (Application Id),
 * fnd:DOC_SEQ_METHOD_CODE (Method Code),
 * fnd:DOC_SEQ_CATEGORY_CODE (Category Code),
 * fnd:DOC_SEQ_DETERMINANT_VALUE (Ledger Id),
 * fnd:DOC_SEQ_DETERMINANT_TYPE
 * fnd:DOC_SEQ_TXN_DATE (Transaction Date)
 * Application Id, Method Code, Category Code, Ledger Id and Transaction Code,
 * should be populated with valid Groovy expressions.
 * The Groovy expressions when evaluated should return a Long for Application Id,
 * and Ledger Id, String for Method Code and Category Code, Timestamp for Transaction Date fields.
 */
* This method will be invoked by postChanges() method on the entity, when
* posting the data to the database.
* 
* Exception will be raised if document sequence value validation fails.
* 
* Override this if you want a different behavior/way of validating the document
* sequence.
* 
* @param appId Application Id
* @param categoryCode Document Sequence Category Code
* @param detType Determinant Type
* @param sobId Ledger Id to use for this Document Sequence.
* @param methodCode Document sequence Method Code (Automatic ("A"), Manual
* ("M") or null for both modes).
* @param txnDate Document Transaction Date
* @param seqVal Document Sequence Value in Manual Mode
* @param suppressWarn Suppress warning (Y/N/null)
* @param suppressError Suppress Error (valid values are Y/N/null)
* @see #postChanges
* /
public void validateDocSequence(Long appId, String categoryCode, Long sobId,
String methodCode, Timestamp txnDate, Long seqVal, String suppressWarn, String
suppressError)
{...}

11.6.1.1 What Happens with Document Sequences at Design Time

Oracle Fusion Middleware extensions provide the ability to identify if an entity
attribute needs a document sequence. This is accomplished by setting Document
Sequence to true in the entity attribute's Property Inspector window, as shown in
Figure 11-1.

Figure 11-1  Property Inspector: Document Sequence
By default, the entity attribute property is not set because it is not a document sequence field.

Fusion Middleware extensions also capture the additional metadata (as Groovy expressions) needed to generate a document sequence. Table 11-4 lists the metadata fields in the Property Inspector window and their descriptions.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Id</td>
<td>The application ID. The Groovy expression should return an object of type Long.</td>
</tr>
<tr>
<td>Category Code</td>
<td>The document sequence category code. The Groovy expression should return an object of type String.</td>
</tr>
<tr>
<td>Method Code</td>
<td>The document sequence method code. Select from the following: &quot;A&quot; (Automatic), &quot;M&quot; (Manual), or null (both modes). The Groovy expression should return an object of type String.</td>
</tr>
<tr>
<td>Determinant Type</td>
<td>The determinant type to use for this document sequence. The Groovy expression should return an object of type String.</td>
</tr>
<tr>
<td>Determinant Value</td>
<td>The determinant value to use for this document sequence. The Groovy expression should return an object of type Long.</td>
</tr>
<tr>
<td>Transaction Date</td>
<td>The document transaction date. The Groovy expression should return an object of type Timestamp.</td>
</tr>
</tbody>
</table>

11.6.1.2 What Happens with Document Sequences at Runtime

Based on the design time setting and additional metadata, Fusion Middleware extensions invoke document sequencing APIs and populate the attribute with a document sequence (in automatic mode) and validate the document sequence (in manual mode) in the postChanges() phase of the entity in ADF Business Components lifecycle. Document sequence processing is done at this phase so that the document sequence generation can be delayed as much as possible when in automatic mode. This is to avoid the potential wasting of document sequence if generated earlier.

If Document Sequence is not set or is set to false, nothing is done. If Document Sequence is set to true, Fusion Middleware extensions populate the entity attribute with a document sequence value if the method code is automatic.

This is accomplished by doing the following.

- Evaluating the Groovy expressions corresponding to the additional required metadata
- Invoking the document sequence PL/SQL APIs to do one of the following:
  - Generate a document sequence value when invoked in automatic mode
  - Validate the document sequence value when invoked in manual mode

11.7 About Managing PL/SQL APIs

Document sequence public PL/SQL APIs can be found in the FND_SEQNUM package. The package can be used to retrieve information about document sequences and assignments, create new document sequences or assignments, and to verify or retrieve the next sequence value for a particular document sequence assignment.
Sample APIs are shown in the examples that follow. For complete documentation, see comments in the package header.

**Example 11-2 Define a New Document Sequence**

```plsql
declare
    ret number;
begin
    -- Define a new document sequence
    ret := fnd_seqnum.define_doc_seq(
        app_id      => 222,               -- Application ID
        docseq_name => 'MY_DOC_SEQUENCE', -- Unique Doc_Seq Name
        docseq_type => 'A',               -- Sequence Type
        ('A'=automatic,'G'=gapless,'M'=manual)
        msg_flag    => 'Y',               -- Message Flag
        init_value  => 1,                 -- Initial sequence value
        start_date  => sysdate,           -- Effective Start date
        end_date    => null,              -- Effective End date
        det_type    => 'BU');             -- Determinant Type value
    if (ret <> FND_SEQNUM.SEQSUCC) then
        dbms_output.put_line('Fail: '||to_char(ret));
    end if;
end;
```

**Example 11-3 Retrieve the Next Sequence Value for an Automatic Sequence**

```plsql
declare
    ret number;
    docseq_val number;
    docseq_id number;
begin
    -- Retrieve the next sequence value for an Automatic sequence
    ret := fnd_seqnum.get_seq_val(
        app_id      => 222,               -- Application ID
        cat_code    => 'MY_CAT',          -- Category code
        sob_id      => 12345,             -- Determinant value
        met_code    => 'A',               -- Method Code ('A'=automatic/batch, 'M'=manual)
        trx_date    => sysdate,           -- Transaction date
        seq_val     => docseq_val,        -- Doc Seq value (output value for automatic)
        docseq_id   => docseq_id);        -- Doc Seq ID (output)
    if (ret <> FND_SEQNUM.SEQSUCC) then
        dbms_output.put_line('Fail: '||to_char(ret));
    else
        dbms_output.put_line('Next sequence value is '||to_char(docseq_val));
    end if;
end;
```

**Example 11-4 Verify a Sequence Value for a Manual Sequence**

```plsql
declare
    ret number;
    docseq_val number;
    docseq_id number;
begin
    -- Verify a sequence value for a Manual sequence
    docseq_val := 54321;
    ret := fnd_seqnum.get_seq_val(
        app_id      => 222,               -- Application ID
        cat_code    => 'MY_CAT',          -- Category code
        sob_id      => 12345,             -- Determinant value
        docseq_val  => docseq_val,        -- Doc Seq value (output value for automatic)
        docseq_id   => docseq_id);        -- Doc Seq ID (output)
end;
```
Example 11-5  Add Determinant Type

```sql
function define_doc_seq (app_id in number, mod_id in varchar2, docseq_name in fnd_document_sequences.name%TYPE, docseq_type in fnd_document_sequences.type%TYPE, msg_flag in fnd_document_sequences.message_flag%TYPE, init_value in fnd_document_sequences.initial_value%TYPE, p_startDate in date, p_endDate in date default NULL, det_type in fnd_document_sequences.determinant_type%TYPE default NULL, p_audit in boolean default true) return number;
```

Example 11-6  Enable Transaction Date Validation

```sql
function get_seq_val_date_enforced (app_id in number, cat_code in varchar2, det_type in varchar2, sob_id in number, met_code in char, trx_date in date, seq_val in out nocopy number, docseq_id out nocopy number, latest_date_used out date, suppress_error in char default 'N', suppress_warn in char default 'N') return number;
```

11.8 About Integrating Document Sequence Task Flows into Oracle Fusion Functional Setup Manager

Every Oracle application registers task flows with a product called Oracle Fusion Functional Setup Manager. Functional Setup Manager provides a single, unified user interface that allows customers and implementers to configure all Oracle applications by defining custom configuration templates or tasks based on their business needs.

The Functional Setup Manager UI enables customers and implementers to select the business processes or products that they want to implement. For example, an Accounts Payable application can register a setup activity like "Create Invoice" with Functional Setup Manager. After you define a category, for example, "Invoices," in the Categories task flow, document sequence task flows then provide the mechanism that
allows you to define a sequence and specify its properties. You then can assign the sequence to the "Invoices" category in the Assignments region in the same flow.

Table 11-5 lists the task flows and their parameters.

| Table 11-5  Task Flow Parameters |
|-----------------|-----------------|-----------------|-----------------|-----------------|
| Task Flow Name  | Task Flow XML   | Parameters Passed | Behavior                  | Comments               |
| Manage Document Sequence Categories | /WEB-INF/oracle/apps/fnd/applcore/docseq/ui/flow/ManageDocSeqCategoriesTF.xm#ManageDocSeqCategoriesTF | mode='search' [moduleType] [moduleKey] mode='edit' applicationId code [pageTitle] | Allows you to create and edit document sequence categories. Search mode allows you to search and edit categories. The moduleType and moduleKey parameters are optional, and if passed restrict the categories that can be queried and edited. Edit mode allows you to query and edit a single category. The applicationId and code parameters are required, and specify the category to edit. | Search and edit document sequence categories. |
| Manage Document Sequences | /WEB-INF/oracle/apps/fnd/applcore/docseq/ui/flow/ManageDocSequencesTF.xml#ManageDocSequencesTF | mode='search' [moduleType] [moduleKey] mode='edit' name [pageTitle] | Allows you to search and edit document sequences, and assignments of those sequences. Search mode allows you to query and edit sequences. The moduleType and moduleKey parameters are optional; if passed they restrict the sequences that can be queried. Edit mode allows you to edit a single document sequence and its assignments. The name parameter is required, and specifies the sequence to edit. | Search and edit document sequences (and the assignments belonging to a sequence). |

For information about implementing your specific product family, do the following:

1. Access the Oracle Fusion Applications library.
2. See the Implementing Common Features guides for your product family.
This chapter describes how to implement Audit Reports in Oracle Fusion Applications.

This chapter contains the following sections:

- **Introduction to Audit Reports**
- **Implementing Audit Reports in the Framework**
- **Using Audit Reports**

### 12.1 Introduction to Audit Reports

Audit Reports creates a history of the changes that have been made to data in Oracle Fusion Applications. Audit Reports includes information such as what operation was performed on an item, when it was performed, and how the value was changed. The audit information is logged without any interaction from the end user.

The auditing solution brings together multiple underlying auditing frameworks, such as Oracle Fusion Middleware auditing that is based on an Oracle Platform Security Services (OPSS) framework, and Oracle Fusion Applications auditing that is based on an Oracle Fusion Middleware Extensions for Applications framework. The OPSS framework captures audit information through events that applications raise during their processing. The Applications Core framework creates a "shadow" table to record all operations on the base table.

### 12.2 Implement Audit Reports in the Oracle Fusion Applications Framework

Database auditing involves observing a database to be aware of the actions of database users. Audit setup provides a user interface mechanism for administrators to choose and select the objects and the specific attributes that need to be audited. When an object has been set to be audited using the audit setup user interface, any Data Manipulation Language (DML) actions on this object, that is, its underlying database schema objects, are captured and stored for reporting purposes.

The Audit Setup UI page is a task flow that is made available through the Oracle Fusion Middleware Extensions for Applications library. This task flow is available in Functional Setup Manager and can be included in an application that is accessible by the administrator. The UI consists of an Applications Tree Table that holds the structure of the auditable application modules, the children application modules, if present, and all the auditable view objects and their children. The administrator selects the view object on which to start auditing and in the applications table components below the tree table, the administrator chooses to add the attributes that need to be audited.
You also can set up the Fusion Middleware auditing as described in How to Set Up Auditing.

Metadata

Two database schema tables, FND_AUDIT_WEBAPP_AM and FND_AUDIT_ATTRIBUTES, hold metadata information that is required for Auditing.

- FND_AUDIT_WEBAPP_AM captures the list of application modules for an application.

<table>
<thead>
<tr>
<th>WEBAPP</th>
<th>APPLICATION_MODULE</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2(80)</td>
<td>VARCHAR2(240)</td>
</tr>
</tbody>
</table>

- FND_AUDIT_ATTRIBUTES table data is populated during runtime when view objects are configured using the Audit Setup UI. This is an internal table and should not be changed.

<table>
<thead>
<tr>
<th>WEBAPP</th>
<th>VIEW_OBJE</th>
<th>VIEW_ATRI</th>
<th>ENABLED_F</th>
<th>TABLE_NAME</th>
<th>COLUMN_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR2(80)</td>
<td>VARCHAR2(240)</td>
<td>VARCHAR2(80)</td>
<td>VARCHAR2(1)</td>
<td>VARCHAR2(30)</td>
<td>VARCHAR2(30)</td>
</tr>
</tbody>
</table>

12.2.1 How to Set Metadata

This section covers the steps needed to populate and define metadata needed to run the Audit task flows.

To populate and define metadata:

1. Make sure every application has the webApp defined and set. This is done by populating the Oracle Fusion Applications Functional Core (ASK) deployment tables. These tables are populated at deployment time through Oracle Fusion Functional Setup Manager tasks.

   The webApp value is your MODULE_SHORT_NAME from the ASK_MODULES table.

2. Determine, for a given module, what are the applications modules that are to be registered in the metadata table (fnd_audit_webapp_am) for audit purposes.

   You can use the AuditServiceAM to register these applications modules. This service allows data to be extracted and uploaded.

   To use the AuditServiceAM, follow these steps:

   - Right-click your Model project.
   - Select Seed Date > Import/Export.
   - Select AuditServiceAM in the wizard that is displayed. If it is not available, make sure to import it to your project.
   - Click OK.

   If you do not have access to, or know how to use the AuditServiceAM, you can run this SQL example (using appropriate values).
describe fnd_audit_webapp_am;
insert into fnd_audit_webapp_am values ('ApplCoreCRMDemo', 'oracle.apps.fnd.appcore.crmdemo.model.CrmDemoAM', sysdate, 'pjl', sysdate, 'pjl', 'pjl', 1);
commit;
select * from fnd_audit_webapp_am;

3. Enable Audit on the Base Table.

You need to create a shadow table for each base table that is being audited. To create the shadow table:

- Open the .table file in JDeveloper.
- Right-click this object and select Properties.
- In the wizard that is displayed, select User Properties.
- In the list that is displayed on the right, scroll down to select the property Enable Audit and set its value to Y, as shown in Figure 12-1.

Figure 12-1 Setting Audit on a Table

- Save your changes.
- A new .table file will be created. This table will have an underscore (_) at the end of the name.
- Check in this new file.

Note: If your base table is enabled for multi tenant, make the same changes for the shadow table.

4. Set the Shadow Table Name on the entity object.

When a shadow table has been created, set this name on your base table EO.xml file as a schema-based property.

Property Name - "FND_AUDIT_TABLENAME"
Property Value - Shadow table name created in step 4.

<Properties>
  <SchemaBasedProperties>
5. Determine which business objects of each application module may be audited. When the business objects are identified, set a non-translatable schema-based property, named FND_AUDIT_ENABLED, on it with Name="Auditable" and Value="Y". Only when this property is set will the business object be shown in the Audit Setup UI for auditing. This follows an opt-in policy.

6. Determine which attributes of each business object you do want to audit and which ones you do not want to audit. By default:

- Attributes with a schema-based property called FND_AUDIT_ATTR_ENABLED and Value="false" will not be shown in the attribute selection dialog (see Figure 12-4).

- Attributes that are history columns are not shown (for example: Created By, Creation Date, Last Updated By)

- Attributes with a schema-based property called FND_AUDIT_ATTR_ENABLED and Value="true" are on by default if the view object is audited. In the attribute selection dialog (see Figure 12-4), these attributes will be displayed on the right (selected) side of the shuttle.

- If no value is set, the attribute will be available for selection in the attribute selection dialog (see Figure 12-4).
7. Define user-friendly names for application modules, business objects and attributes.

To have a user-friendly display name for all application modules, business objects and attribute names, the label property for each must be set. For application modules and business objects, you should add a tooltip, since the tooltip will be shown in the Audit Setup UI and also will give the administrator further information on the business objects and application module. The label and tooltip are available as properties.

The following shows an example for a view object:

```xml
<Properties>
    <SchemaBasedProperties>
        <fnd:FND_AUDIT_ENABLED Value="true"/>
    </SchemaBasedProperties>
    <LABEL ResId="ViewObjectVO_LABEL"/>
    <TOOLTIP ResId="ViewObjectVO_TOOLTIP"/>
</Properties>
</viewAttribute>
```

The following shows an example for an attribute. Set the Label property under the UI Hints in the property inspector for an attribute.

```xml
<ViewAttribute
    Name="TableName"
    PrecisionRule="true"
    EntityAttrName="TableName"
    EntityUsage="FndAuditAttributesEO"
    AliasName="TABLE_NAME">
    <Properties>
        <SchemaBasedProperties>
            <LABEL ResId="TableName_LABEL"/>
        </SchemaBasedProperties>
    </Properties>
</ViewAttribute>
```

8. Enable a User Key.

The User Key is the value that will appear in the Description field in the Audit Reports. See Figure 12-5.

If you need to be able to search based on a key other than the primary key of the view object, then you need to set a schema-based property called FND_AUDIT_USER_KEY on this attribute. This attribute must have a unique value.
and the new property should be set on only one attribute that is not the primary key. If you have a multi-part key, then you should use the one that is the most important as the User Key, and use the context name/value pairs for the others. Although end users may not be able to search down to just one item, they can see the context values to determine the ones in which they are interested.

```xml
<ViewAttribute
    Name="TableName"
    IsUnique="true"
    IsNotNull="true"
    PrecisionRule="true"
    EntityAttrName="TableName"
    EntityUsage="FndTablesEO"
    AliasName="TABLE_NAME">
  <Properties>
    <SchemaBasedProperties>
      <fnd:FND_AUDIT_USER_KEY Value="true"/>
    </SchemaBasedProperties>
  </Properties>
</ViewAttribute>
```

9. Configure lookups on the User Key attribute.

Foreign Key lookup can be configured on the User Key attribute. To do so, configure the User Key attribute and then configure the Foreign Key lookup. In addition, the annotation listed below needs to be mentioned at View Object level to derive the User Key from the lookup value (while a Description search is performed).

The user key can be something other than a straight value. This is accomplished by allowing more tags to be used with the `FND_AUDIT_USER_KEY` tag:

```xml
  <fnd:FND_AUDIT_USER_KEY Value="true"/>
  <fnd:FND_AUDIT_FK_VIEWOBJECT Value=" "/>
  <fnd:FND_AUDIT_FK_DISPLAYATTR Value=" "/>
  <fnd:FND_AUDIT_FK_WHEREATTR Value=" "/>
  <fnd:FND_AUDIT_FK_WHEREATTR_ADDL Value=" "/>
  <fnd:FND_AUDIT_VALUE_ADDL Value=" "/>
  <fnd:FND_AUDIT_USERKEY_LKP_COND Value=" "/> <!-- This attribute must be used if the FK tags are used. -->
```

Example

Assume the `PersonId` attribute is defined as the User Key and the Foreign Key lookup is defined on the User Key:

```xml
  <fnd:FND_AUDIT_USER_KEY Value="true"/>
  <fnd:FND_AUDIT_FK_VIEWOBJECT Value="abc.PersonVO"/>
  <fnd:FND_AUDIT_FK_DISPLAYATTR Value="PersonName"/>
  <fnd:FND_AUDIT_FK_WHEREATTR Value="PersonId"/>
  <fnd:FND_AUDIT_USERKEY_LKP_COND Value="SELECT PERSON_ID FROM PERSONS WHERE PERSON_NAME LIKE ?"/>
```

10. Configure custom objects.

For custom business objects, identify the application module to which the custom business objects belong and add this to the metadata table as defined in step 5. Define the `FND_AUDIT_ENABLED` property as shown in step 4 on the custom
business objects. If entity objects also are generated at a customer instance, set the property mentioned in step 4.

11. If needed, configure the attribute lookup based on the FND_LOOKUPS table.

For columns that show raw data of lookup codes instead of the meaning (such as "A034" instead of "washing machine"), you can specify annotation across the attribute specifying its lookup type. Audit Report internally queries the FND_LOOKUPS table for that lookup_type to show the column meaning. The schema-based property to use is FND_AUDIT_LOOKUP_TYPE as shown below:

```xml
<ViewAttribute
  Name="ProblemDescription" IsNotNull="true" PrecisionRule="true" EntityAttrName="ProblemDescription"
  EntityUsage="ServiceRequestsEO" AliasName="PROBLEM_DESCRIPTION">&#160;&#160;
  <Properties>
    <SchemaBasedProperties>
      <fnd:FND_AUDIT_LOOKUP_TYPE Value="AHL_PLANNING_VISIT_TYPE"/>
    </SchemaBasedProperties>
  </Properties>
</ViewAttribute>
```

12. If needed, configure context descriptive attributes to show context attributes that define the context of a changed record.

A maximum of 3 context descriptive attributes can be defined for a given view object. If context attributes were defined for both parent and child view objects, the report will show parent context information for parent records, and child context information for child records.

Each context attribute must be defined with an annotation similar to that shown in the example but that has FND_AUDIT_CONTEXT2 and FND_AUDIT_CONTEXT3 as its prefix.

If you need an additional WHERE condition to derive the context attribute value, you can use the _ADDL annotations, as shown in the example.

```xml
<ViewObject ...>
  <Properties>
    <CustomProperties>
      <Property Name="AUDIT_CONTEXT1_VIEWOBJECT" Value=""/> <!--fully qualified view object name, optional for same view -->
      <Property Name="AUDIT_CONTEXT1_DISPLAYATTR" Value=""/> <!--attribute of lookup view to display-->
      <Property Name="AUDIT_CONTEXT1_WHEREATTR" Value=""/> <!--attribute of lookup view to apply WHERE condition-->
      <Property Name="AUDIT_CONTEXT1_WHEREATTR_ADDL" Value=""/> <!--attribute of base view for WHERE condition key-->
      <!-- composite key specification - start-->
      <Property Name="AUDIT_CONTEXT1_WHEREATTR_ADDL" Value=""/> <!--attribute of lookup view to apply additional WHERE condition-->
      <!-- either <Property Name="AUDIT_CONTEXT1_VALUE_ADDL" Value=""/> <!-- for hardcoded value-->
      or <Property Name="AUDIT_CONTEXT1_KEYATTR_ADDL" Value=""/> <!-- attribute of base view for additional key-->
      <!-- composite key specification - end-->
    </CustomProperties>
  </Properties>
</ViewObject>
```
Example Use Case

You want to specify Product as one of the context attributes for the Revenue view objects. But Product is composed of a composite key: InventoryItemId and InventoryOrgId.

The specification will appear similar to:

```xml
<ViewObject name="Revenue"...>
  <Properties>
    <CustomProperties>
      <Property Name="AUDIT_CONTEXT1_VIEWOBJECT" Value="com.oracle...
EgpSystemItemsVO"/>
      <Property Name="AUDIT_CONTEXT1_DISPLAYATTR" Value="Description"/>
      <Property Name="AUDITCONTEXT1_WHEREATTR" Value="InventoryOrgId"/>
      <!-- composite key specification - start-->
      <Property Name="AUDITCONTEXT1_WHEREATTR_ADDL" Value="InventoryItemId"/>
      <Property Name="AUDITCONTEXT1_KEYATTR_ADDL" Value="InventoryItemId"/>
      <!-- for additional key attr -->
      <!-- composite key specification - end-->
    </CustomProperties>
  </Properties>
</ViewObject>
```

The SQL statement that will be generated will be:

```sql
SELECT Description from <table used by EgpSystemItemsVO> where (InventoryOrgId = Revenue.InventoryOrgId) AND (InventoryItemId = Revenue.InventoryItemId)
```

13. If needed, configure attribute lookup based on Foreign Key tables.

This works the same as Configure Attribute Lookup Based on the FND_LOOKUPS table, but is not limited to the FND_LOOKUPS table.

```xml
<ViewObject ...
  <ViewAttribute
    Name="OWNER"IsNotNull="true"PrecisionRule="true"EntityAttrName="Owner"EntityUsage="ServiceRequestsEO"AliasName="Owner">
    <Properties>
      <SchemaBasedProperties>
        <fnd:FND_AUDIT_FK_VIEWOBJECT Value=""/>
        <fnd:FND_AUDIT_FK_DISPLAYATTR Value=""/>
        <fnd:FND_AUDIT_FK_WHEREATTR Value=""/>
      </SchemaBasedProperties>
    </Properties>
</ViewAttribute>
</ViewObject>
```

14. If needed, define View Criteria on Business Objects to filter the display of view objects. For instance, a table might be shared by three applications, and you need to filter based on only one of the applications.

Simple query-based View Criteria:

```xml
<ViewObject ...
  <Properties>
    <SchemaBasedProperties>
      <fnd:FND_AUDIT_VIEWCRITERIA_DBOBJECT Value=""/>
    </Properties>
</ViewObject>
```
Complex query-based View Criteria:

15. If needed, force writing attributes to the Shadow Table.

The `FND_AUDIT_FORCE_WRITE` property co-exists with the normal audit properties.

Product teams may want to write certain attributes to the shadow table all the time, even if the attribute is not enabled for auditing. This could be because Virtual Private Datasource policy is set up on the attribute, or the attribute participated in a WHERE condition. This sample code shows how to enable force writing of the attributes:

```
<ViewAttribute
    Name="TableName"
    IsUnique="true"
    IsNotNull="true"
    PrecisionRule="true"
    EntityAttrName="TableName"
    EntityUsage="FndTablesEO"
    AliasName="TABLE_NAME">
    <Properties>
        <SchemaBasedProperties>
            <fnd:FND_AUDIT_FORCE_WRITE Value="true"/>
        </SchemaBasedProperties>
    </Properties>
</ViewAttribute>
```

- If the property value is set to "true," and the attribute is not enabled for audit, it will not appear in the audit reports (unless it also is defined as the user key, context attribute, and so on).

- If the property value is set to "true," and the attribute is enabled for audit, it will only appear when the value changes.

```
<Properties>
    <SchemaBasedProperties>
        <fnd:FND_AUDIT_FORCE_WRITE Value="true"/>
    </SchemaBasedProperties>
</Properties>
```

Note that when a search is conducted on a view object, then the search result will include rows from even the child view objects.
16. If needed, exclude a view object from a search in the Audit Report.

The Audit Reports UI contains a search query panel in which a user can select a particular Product from the Product LOV and then filter the search further by selecting a view object from the Business Object LOV.

The fnd:FND_AUDIT_REPORT_EXCLUDE audit annotation provides the ability to exclude a particular view object from the Business Object LOV, even if it may be marked as auditable. This is required for use cases in which a view object without the context of its parent will not make sense, so developers can expose the parent object and hide the children.

```
<ViewObject
xmlns="http://xmlns.oracle.com/bc4j"
xmlns:fnd="http://xmlns.oracle.com/apps/entityview"
...
</Properties>
</ViewObject>
```

17. If needed, configure custom display names for Context attribute names.

The custom display name of the Context attribute name in an Audit Report can be configured by using the annotation fnd:FND_AUDIT_CONTEXT1_LABEL_ID. The value provided to this annotation must be a resource ID in the resource bundle of the view object in which these context annotations are defined. If the annotation is not used, then the display name of the context view attribute itself is displayed in the Audit Report.

```
<ViewObject ...
<Properties>
  <SchemaBasedProperties>
    <fnd:FND_AUDIT_CONTEXT1_VIEWOBJECT Value=""/>
    <fnd:FND_AUDIT_CONTEXT1_DISPLAYATTR Value=""/>
    <fnd:FND_AUDIT_CONTEXT1_WHEREATTR Value=""/>
    <fnd:FND_AUDIT_CONTEXT1_KEYATTR Value=""/>
    <fnd:FND_AUDIT_CONTEXT1_LABEL_ID Value=""/>
  </SchemaBasedProperties>
</Properties>
</ViewObject>
```

The custom display name can be configured to each Context attribute by using the FND_AUDIT_CONTEXT2_LABEL_ID and FND_AUDIT_CONTEXT3_LABEL_ID annotations.

18. If needed, configure lookups on the User Key attribute.

A Foreign Key lookup can be configured on a User Key attribute. Configure a User Key attribute as described in step 8. Then configure a Foreign Key lookup as described in step 13. Include this annotation at the view object level to derive a User Key from the lookup value while a Description search is performed.

```
<ViewObject ...
<Properties>
  <SchemaBasedProperties>
    ....
  </Properties>
</ViewObject>
```
Example

Assume that the PersonId attribute is defined as a User Key, and Foreign Key lookup is defined on the User Key, as shown here:

```xml
<viewObject ...>
<properties>
    <schemaBasedProperties>
        ....
        <fnd:FND_AUDIT_USERKEY_LKP_COND Value="SELECT PERSON_ID FROM PERSONS WHERE PERSON_NAME LIKE ?"/>
    </schemaBasedProperties>
</properties>
</viewObject>
```

19. If needed, you can hide a view object under an application module in the Audit Setup UI

If the view object contains the audit annotation `<fnd:FND_AUDIT_ENABLED Value="true"/>`, it will be displayed in the Setup UI. However, there are cases in which the view object that is owned by a different team is present under the application module and the user is not interested in seeing it under the application module in the Setup UI.

For example, when Oracle Fusion CRM creates custom objects, it also creates an Attachment object as its child, by default. However, they do not want to see the Attachment object in the Setup UI. Since the Attachment view object is an FND object and is not owned by Oracle Fusion CRM, there is no way for them to remove the `fnd:FND_AUDIT_ENABLED` annotation from it.

The annotation shown here can be specified in the application module which is registered for audit in the `fnd_audit_webapp_am` table. The user is expected to provide a comma-separated list of all view object names that is present in the Data Model of the application module and needs to be hidden in the Audit Setup UI. With this, in place the listed view objects and their children, if any, will be hidden.

```xml
<appModule ...>
    <properties>
        ...
        <customProperties>
            <property Name="fnd:FND_AUDIT_SETUP_HIDE_VIEWOBJECT_LIST" Value = 'oracle.apps.fnd.applcore.audit.setup.view.FndTablespacesVO'/>
        </customProperties>
    </properties>
</appModule>
```

20. If needed, support audit-specific Data Security view criteria in a view object
Audit considers the Data Security privilege for Business Object instances when specified in application modules (using view criteria in the format of FNDDS_privilegeNameobjectName_objectAlias) or if specified at the Entity level by enabling the "read" privilege in the Security section.

If, for any reason, this cannot be done, audit supports the annotation shown here that allows the Data Security privilege to be specified in the view object. Audit History will consider this view object-level privilege as first priority.

```xml
<ViewObject ...
  <Properties>
    <SchemaBasedProperties>
      ....
      <fnd:FND_AUDIT_DS_VIEWCRITERIA
        Value="FNDDS__read__FND_CRM_CASES__QRLT,FNDDS__read__FND_CRM_CASES__CasesEO"/>
    </SchemaBasedProperties>
  </Properties>
</ViewObject>
```

21. If needed, there is audit support for PL/SQL-based Data Manipulation Language (DML) operations

Audit maintains a history of changes done to business objects in Oracle Fusion Applications.

During business transaction flows based on ADF Business Components, audited attribute values are stored in shadow tables. Audit automatically extends product transactions without needing to change product team code or business flows. For transaction flows using PL/SQL, data is not automatically saved to the shadow table. The changes shown here must be made in the PL/SQL procedures that write to the base tables.

**API Signature**

```
-- Type tr_data will contain data for only one of the variables based on the column's data type.
-- If the column is varchar, varchar2_value needs to be set.

TYPE tr_data IS RECORD(
  varchar2_value VARCHAR2(4000),
  number_value NUMBER,
  date_value DATE,
  timestamp_value TIMESTAMP);

TYPE tr_column_data IS RECORD(
  column_name VARCHAR2(30),
  column_type VARCHAR2(30),
  old_data tr_data,
  new_data tr_data);

TYPE tt_columns_data is table of tr_column_data INDEX BY BINARY_INTEGER;

-- get_audited_columns function returns the list of columns enabled for auditing for the passed in base table.
-- Parameters
-- p_base_table_name - pass the base table name on which the DML Operation occurred
```

FUNCTION get_audited_columns (p_base_table_name IN VARCHAR2) RETURN
  tt_column_names;

-------------------------------------------------------------------------------
-- Create_Audit_Row method writes to the audit shadow table
-- Parameters
-- p_base_table_name- pass the base table name on which the DML Operation occurred
-- p_event_type     - pass "INSERT", "UPDATE", or "DELETE" based on the DML
  operation on base table
-- p_audit_data     - pass a table structure with each row(tr_column_data) contain the column,
  its type and old and new value. All the columns returned from get_audited_columns needs to be passed.
-- Additionally all the who columns (created_by, creation_date, last_update_date, last_updated_by, last_update_login needs to be passed. In case of INSERT, only new data needs to be passed and in case of DELETE, only old data. In case of UPDATE, pass both the old and new value even for attributes that are not changed.
----------------------------------------------------------------------------------

-----------------------------------------------------------------------------

PROCEDURE create_audit_row(
  p_base_table_name IN VARCHAR2,
  p_event_type IN VARCHAR2,
  p_audit_data IN tt_columns_data);

Coding Steps

• Check if the table is enabled for Auditing

  FND_AUDIT_UTIL.is_table_audited(
    p_base_table_name in varchar2(30)  // input parameter - base table name
  ) return varchar2(5)               // returns "true" or "false"

  If the API returns false, do nothing.

• If the table is audited and the DML type is INSERT for the base table, call the
  PL/SQL API, passing in all the values as new values.

    declare
      tt FND_AUDIT_UTIL.tt_columns_data;
      p_base_table_name varchar2(30);

    begin
      p_base_table_name := 'FND_LOOKUP_TYPES';
      if (FND_AUDIT_UTIL.is_table_audited(p_base_table_name) = 'false') then
        return;
      end if;

      tt(1).column_name := 'CUSTOMIZATION_LEVEL';
      tt(1).column_type := 'VARCHAR2';
      tt(1).new_data.varchar2_value := 'U';

      tt(2).column_name := 'ENTERPRISE_ID';
      tt(2).column_type := 'NUMBER';
      tt(2).new_data.NUMBER_value := '0';

      tt(3).column_name := 'LOOKUP_TYPE';
      tt(3).column_type := 'VARCHAR2';
      tt(3).new_data.varchar2_value := 'TEST11';
FND_AUDIT_UTIL.CREATE_AUDIT_ROW(
   p_base_table_name,
   'INSERT',
   tt);
end;

• If the table is audited and the DML type is UPDATE, call the PL/SQL API to get a list of audit-enabled attributes.

```
TYPE tt_column_names  IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;
FND_AUDIT_UTIL.get_audited_columns(
   p_base_table_name in varchar2(30)  // input parameter - base table name
) return tt_column_names
```

If any column that is updated in the current transaction is in the list returned from get_audited_columns, get the value for all the other columns in the table and call the PL/SQL API. Note that you need to pass in values (old and new) for all the columns returned from the get_audited_columns() function, and also for all the who columns, even if the values have not changed.

```
declare
   tt FND_AUDIT_UTIL.tt_columns_data;
   p_base_table_name varchar2(30);
   col_list  FND_AUDIT_UTIL.tt_column_names;
   l_col_change_found varchar2(30);
begin
   p_base_table_name := 'FND_LOOKUP_TYPES';
   l_col_change_found := 'false';
   if (FND_AUDIT_UTIL.is_table_audited(p_base_table_name) = 'false') then
      return;
   end if;
   col_list := FND_AUDIT_UTIL.get_audited_columns(p_base_table_name);
```
-- assuming here that only customization_level has changed
FOR i IN 1 .. col_list.count LOOP
  IF (col_list(i) = 'CUSTOMIZATION_LEVEL') THEN
    l_col_change_found := 'true';
  END IF;
END LOOP;

-- if you don't have all the values, query the rest of the values from your
base table
-- then pass the same value for old and new value.
-- if the old or new value is blank, audit code will assume that the value
has been cleared out.

if (l_col_change_found :='true') THEN
  tt(1).column_name := 'CUSTOMIZATION_LEVEL';
  tt(1).column_type := 'VARCHAR2';
  tt(1).old_data.varchar2_value := 'U';
  tt(1).new_data.varchar2_value := 'S';
  tt(2).column_name := 'ENTERPRISE_ID';
  tt(2).column_type := 'NUMBER';
  tt(2).old_data.NUMBER_value := '0';
  tt(2).new_data.NUMBER_value := '0';
  tt(3).column_name := 'LOOKUP_TYPE';
  tt(3).column_type := 'VARCHAR2';
  tt(3).old_data.varchar2_value := 'TEST11';
  tt(3).new_data.varchar2_value := 'TEST11';
  tt(4).column_name := 'MODULE_ID';
  tt(4).column_type := 'VARCHAR2';
  tt(4).old_data.varchar2_value := '40B3FA7250D19380E040449823C67A1A';
  tt(4).new_data.varchar2_value := '40B3FA7250D19380E040449823C67A1A';
  tt(5).column_name := 'VIEW_APPLICATION_ID';
  tt(5).column_type := 'NUMBER';
  tt(5).old_data.NUMBER_value := '0';
  tt(5).new_data.NUMBER_value := '0';
  tt(6).column_name := 'CREATED_BY';
  tt(6).column_type := 'VARCHAR2';
  tt(6).old_data.varchar2_value := 'ADMIN';
  tt(6).new_data.varchar2_value := 'ADMIN';
  tt(7).column_name := 'LAST_UPDATED_BY';
  tt(7).column_type := 'VARCHAR2';
  tt(7).old_data.varchar2_value := 'ADMIN';
  tt(7).new_data.varchar2_value := 'APPLICATION_ADMINISTRATOR';
  tt(8).column_name := 'CREATION_DATE';
  tt(8).column_type := 'TIMESTAMP';
  tt(8).old_data.timestamp_value := sysdate -10;
  tt(8).new_data.timestamp_value := sysdate-10;
  tt(9).column_name := 'LAST_UPDATE_DATE';
  tt(9).column_type := 'TIMESTAMP';
  tt(9).old_data.timestamp_value := sysdate -10;
  tt(9).new_data.timestamp_value := sysdate;

FND_AUDIT_UTIL.CREATE_AUDIT_ROW(
  p_base_table_name,
• If the table is audited and the DML type is DELETE for the base table, call PL/SQL API, passing in all the values (as old values) from the base table.

```sql
DECLARE
    tt FND_AUDIT_UTIL.tt_columns_data;
    p_base_table_name varchar2(30);
BEGIN
    p_base_table_name := 'FND_LOOKUP_TYPES';
    IF (FND_AUDIT_UTIL.is_table_audited(p_base_table_name) = 'false') THEN
        RETURN;
    END IF;

    tt(1).column_name := 'CUSTOMIZATION_LEVEL';
    tt(1).column_type := 'VARCHAR2';
    tt(1).old_data.varchar2_value := 'S';

    tt(2).column_name := 'ENTERPRISE_ID';
    tt(2).column_type := 'NUMBER';
    tt(2).old_data.NUMBER_value := '0';

    tt(3).column_name := 'LOOKUP_TYPE';
    tt(3).column_type := 'VARCHAR2';
    tt(3).old_data.varchar2_value := 'TEST11';

    tt(4).column_name := 'MODULE_ID';
    tt(4).column_type := 'VARCHAR2';
    tt(4).old_data.varchar2_value := '40B3FA7250D19380E040449823C67A1A';

    tt(5).column_name := 'VIEW_APPLICATION_ID';
    tt(5).column_type := 'NUMBER';
    tt(5).old_data.NUMBER_value := '0';

    tt(6).column_name := 'CREATED_BY';
    tt(6).column_type := 'VARCHAR2';
    tt(6).old_data.varchar2_value := 'ADMIN';

    tt(7).column_name := 'LAST_UPDATED_BY';
    tt(7).column_type := 'VARCHAR2';
    tt(7).old_data.varchar2_value := 'APPLICATION_ADMINISTRATOR';

    tt(8).column_name := 'CREATION_DATE';
    tt(8).column_type := 'TIMESTAMP';
    tt(8).old_data.timestamp_value := sysdate -10;

    tt(9).column_name := 'LAST_UPDATE_DATE';
    tt(9).column_type := 'TIMESTAMP';
    tt(9).old_data.timestamp_value := sysdate;

    FND_AUDIT_UTIL.CREATE_AUDIT_ROW(
        p_base_table_name,
        'DELETE',
        tt);
END;
```
12.2.2 How to Set Up Auditing

As with most Oracle Fusion applications, Audit setup is done from Functional Setup Manager. When you start the setup, search in Functional Setup Manager for "Audit." When it is located, select Audit Setup.

A list of Oracle Fusion applications is displayed, as shown in Figure 12-2.

**Figure 12-2  Initial Audit Setup Display**

You will notice that some have only an Audit Level drop-down list that lets you pick Low, Medium or High levels. These are Oracle Fusion Middleware products that generate event-based entries in the table that is read by the Audit system, and the user just sets the granularity that is desired. (Refer to the product documentation and to the online help for a description of how the levels work in each particular product.)

In the example figure, click the **Configure Business Object Attributes** button to display the Configure Business Objects Attributes dialog. When it first displays, the fields will be empty.

To populate the Objects list, open the Application drop-down list and select the name of the application you want to audit. The example shown in Figure 12-3 uses the Applications Core Setup application.
There are three buttons at the top of the Configure Business Object Attributes dialog:

- Click **Save** to save your work and remain in the Audit Setup dialog.
- Click **Save and Close** to save your work and exit the Audit Setup dialog.
- Click **Cancel** to cancel any changes you have made since the last Save action and exit the Audit Setup dialog.

**Objects List**

The Objects list has three columns:

- **Audit**

  There is no limit to the number of levels that can be included in an audit.

  You will notice that items with the same name are displayed and can be selected. The item at the top of the tree is the root item; the other listed items of the same name are reflections of the root item and are not separate. That is, if a view object appears multiple times in the Setup list, there really is only one view object. Selecting or deselecting any instance of that view object will select or deselect all instances.

  **Example:**

  You pick a view object and select five of its attributes to audit and save. If you then choose the same view object, but somewhere else in the setup UI, it will have those five attributes already selected. Now select two more, so that there are seven attributes selected. When you return to the first instance of the view object, it also will show seven attributes selected.

  Similarly, if you delete one of the five selected attributes, all instances of the affected view object will show only four selected attributes.

  Another way of saying this is "last edit wins."

  Any item that is checked in the Audit column will have all its selected attributes audited. Unless the view objects’ Attributes list has been edited, the attributes that will be audited are default attributes.

  If you edit a view object to select specific attributes, the Audit box must still be checked to audit the selected attributes. (You can edit the attributes, but that, by
itself, does not mean they will be included in an audit, unless the Audit box is checked.)

- **Name**
  The names of the application modules and view objects.

- **Description**
  A brief description of the associated application module or view object.

The Objects list has two menus.

- **Actions menu**
  The Actions menu provides the **Synchronize** option.
  Attributes of objects that are being audited are tracked. On a patch or update, or if customized attributes, such as flexfield attributes, have been added, they must be tracked if they are meant to be included in the audit. That is, the attribute was marked as `<fnd:FND_AUDIT_ATTR_ENABLED Value="true"/>`. Synchronize will do this.

  **Note:** Synchronization is carried out by product. That is, it will synchronize all objects in the current product shown in the UI, if those objects have been selected for audit at any time previously.

  Flexfield and custom attributes are always included by default (`<fnd:FND_AUDIT_ATTR_ENABLED Value="true"/>`), so with these you want to be sure to do a synchronization.

- **View menu**
  Use the View menu to select the columns you want to display, and to expand and collapse entries in various combinations.

**Audited Attributes List**

When you click a view object entry in the Objects list, its attributes that will be included in an audit are displayed in the Audited Attributes fields. Note that the name of the selected object is added before the Audited Attributes heading; for example Document Entities: Audited Attributes.

**Note:**

The permitted limit on selected attributes is 20,000 bytes (num_attributes x attribute_size) for a given view object in the Audit Setup UI. This limit is checked during setup. Note that custom text attributes are pre-allocated with 1,500 chars per field. Therefore, it would only take 13 Text fields to consume the available character width.

These are not necessarily all the attributes of the selected view object; they either are the default attributes or the attributes selected during a previous edit. To see a list of all a view object’s attributes, select **Actions > Create**, or click the **Create** icon. A dialog, similar to that shown in Figure 12-4, lists all the attributes and lets you shuttle them between the Available Attributes and the Selected Attributes.
Click **Save and Close** to save your selections. Remember that your selections will not be included in an audit unless the related Audit checkbox also is selected.

### 12.3 Using Audit Reports

Open Audit Reports from the Navigator menu.

The Audit Reports Search page, shown in **Figure 12-5**, is displayed.

**Figure 12-5  Default Audit Reports Display**

To perform a search, a Date value and *either* a User or Product value are required. Use the Date picker to select the date, and the drop-down lists to select a User or a Product or both. The search results can be further fine-tuned by using the Event Type, Business Object and Description fields, and the Include child objects checkbox.

**Note:**

These parameters can be saved and recalled by selecting the user-supplied name in the Saved Search drop-down list. See also "If the search parameter set you are using is one that you expect to reuse, click Save to display the Create Saved Search dialog that has these options:’’
12.3.1 How to Search in Audit Reports

To search for Audit Reports for an Oracle Fusion application, you must supply this information:

- **Date.** Select an operator, such as Equals, Before, After or Between, and click the picker to display a calendar from which to choose a date.

- **User** or **Product** or both.

When you use the drop-down to select a User, you will notice a **Search** link at the bottom of the list. Click it to access all the available Users. The dialog, shown in Figure 12-6, is displayed.

**Figure 12-6  Searching for a User**

You can enter a name or part of a name in the User field and click **Search.** You also can click **Advanced** to add a search operators drop-down list, as shown in Figure 12-7.

**Figure 12-7  User Name Operators**

To select a user, highlight the name in the search results, shown in Figure 12-8, and click **OK.**
**Event Type.** Note that the first drop-down list always is set to Equals. Use the second drop-down list to select any or a combination of the available types. You also can select All.

For Oracle Fusion applications, the Types are Object Data Insert, Object Data Update, and Object Data Delete.

For Oracle Fusion Middleware, the Type selection is dynamic and depends on the selected Product.

**Business Object Type.** Note that this field applies only to Oracle Fusion applications and not to Oracle Fusion Middleware.

**Description.** You can further limit the search results display by entering a string in this field that matches a string in the Description column of the search results.

**Include child objects.** If there is a hierarchy of view objects, such as Opportunity > Revenue Line > Revenue Line Split, and the search is done on Opportunity, it only checks the Opportunity view object for changes. But if this option is checked, it will search all children. So if an Opportunity has five Revenue Lines and each has some Revenue Line Splits, then if anything in any of them changed, they also would be reported.

When the necessary fields are filled-in, click **Search** to display the results in the Search Results table.

---

**Note:**

When a search is performed for broad criteria, such as “Specific Product Before Today” or “All Products Before Today,” Audit Report may take a long time to query Audit History.

To avoid intermittent connection reset errors, *the search time is limited to 5 minutes.* If the History search time exceeds 5 minutes, this message is displayed: “Query run time exceeded the permissible time limit and failed to display results. Refine the search criteria and try again.”

---

**Deleting an Object Referenced by a Foreign Key**

In some instances, resolving a foreign key may fail when an audit row display name is trying to do a foreign key lookup of an object that has been deleted. It then will display the raw value or just leave it blank.
One example of this behavior involves trying to get a foreign key resolution from the same base object. If rows were deleted from an object, there will not be a row in the base table and foreign key resolution may fail.

Another example involves foreign key resolution with a parent table. For instance, given a person object, a person email object may use foreign key resolution to show a person’s name instead of a person ID. If the person’s name has been deleted in the base table, the foreign key resolution for the person email object may fail because there is no row in the corresponding base table.

Click **Reset** to clear all the fields so you can enter new search parameters.

If the search parameter set you are using is one that you expect to reuse, click **Save** to display the Create Saved Search dialog that has these options:

- **Name**
  This option is mandatory.
  This field defaults to showing the name that is displayed in the Saved Search field. However, if the Saved Search displayed name is “My Recent Changes,” the Name field here will show “My Recent Changes copy.”

- **Set as Default**
  Select this option to designate this named set of search parameters as the default instead of My Recent Changes.
  This option can be set and cleared by opening the Saved Search drop-down list and selecting Personalize.

- **Run Automatically**
  Select this option to run this named set of search parameters automatically.
  This option can be set and cleared by opening the Saved Search drop-down list and selecting Personalize.

- **Show in Search List**
  This option is available only from the Saved Search Personalize option, and is active only for the My Recent Changes named search.

### 12.3.1.1 Using the Search Results Table

The Search Results table has these components:

- **Actions menu**
- **Export to Excel icon** that duplicates the function of the Actions menu option of the same name
- **Detach icon**
- **Show Attribute Details checkbox.** This option becomes active when the Business Object Type field is populated and a search has been performed.
  
  When this option is selected, a dropdown list lets you select either All Attributes or Select Attribute.
When this option is selected, three columns are added to the table: Attribute, Old Value and New Value. Attribute contains the attribute name. An Old Value will be shown if there was an old value. New Value shows the current value. A scroll bar will become available at the bottom of the display so you can view all the columns.

Select this option when you want the information for only one specific attribute. When selected, a popup similar to this example is displayed.

You must click the Business Object to populate the Select Attribute list. Click OK to apply the change to the table.

The search results table now shows only results with the selected attribute, plus the Old Value and New Value columns that are shown when the All Attributes option is selected.

- Show Extended Object Identifier Columns checkbox.
When this option is selected, three new pairs of columns are added to the Search Results table: Context Name1 with Context Value1 through Context Name3 with Context Value3. These are additional information to help identify the object. For instance, if you need a 4-part key to uniquely identify an object, the Description could be 1 and these 3 context values could be the remaining.

Note that when a search is conducted on a view object, the search result also will include rows from the child view objects.

**Actions Menu**

The Actions menu has the Export to Excel option that is duplicated with the Export to Excel icon. Select the menu option or click the icon to export all the entries in the Search Results list to an Excel XLS file. When this option is selected, the standard Save prompts will guide you through the process.
Get Started with Your Web Interface

This chapter provides information that you may need before you begin developing your web pages. It introduces the UI Shell page template and UI patterns and features that are available in JDeveloper.

This chapter includes the following sections:

• Introduction to Developing a Web Application
• Oracle Fusion Guidelines, Patterns, and Standards
• Basic Building Blocks
• Introduction to the UI Shell
• Applications UI Patterns and Features

13.1 Introduction to Developing a Web Application

To help you get started with your web interface, this chapter discusses information about how to create a page, what the wizard settings should be, as well as information about patterns, such as UI Shell.

For more information about how to get started with your web interface, see the Oracle Fusion Middleware Developing Web User Interfaces with Oracle ADFFaces guide.

13.2 Oracle Fusion Guidelines, Patterns, and Standards

There are some basic guidelines for defining how an Oracle Fusion application’s web interface is constructed. These guidelines are universally shared by all pages built for Oracle Fusion Applications. There are two types of pages - Dashboards and Work Areas. A dashboard is a collection of information summaries (high-level data views) that enable users to monitor different objects and data within a subdomain or functional area of interest. A Work Area is the complete set of tasks, reports, business intelligence, searches and other content that a user needs to accomplish the tasks associated with a business goal. Depending on the type of page you have, the construction can differ.

For more information about Standards and Guidelines, see Chapter 1, "Getting Started with Oracle Fusion Applications."

13.3 Basic Building Blocks

Dashboards and Work Areas define the basic structure of a page. Oracle Fusion Guidelines, Patterns and Standards (GPS) defines a set of design patterns. Design patterns are common flow or page designs that are used across all product families. By using design patterns in all phases of product development, valuable development time may be spent innovating other areas in the product, consistency is ensured across
the entire enterprise, and users only have to learn the interaction once with the expectation that their experience will be the same in any product they encounter.

Dashboard

There are two types of dashboards: Home Based Dashboards and Transaction Dashboards. A Home Based Dashboard consists of one or more tabs. Each tab is a container for a set of configurable regions displaying content that a user may want to monitor. Transaction Dashboards are built with a specific role in mind. The basic building blocks are the individual regions. Every dashboard can choose which regions it wants to include. In Figure 13-1, these are the Watchlist, Reports and Analytics, Worklist, and Gallery. Each of these is built as Oracle Application Development Framework (Oracle ADF) Bounded Task Flows.

Figure 13-1  Home Based Dashboard Example

Work Area

A Work Area, as shown in Figure 13-2, consists of a Regional Area, a Local Area and a Contextual Area. Each area is intended to have content for a specific purpose.

Figure 13-2  Work Area example

The Regional Area is the collapsible region on the left of the page that contains a column of panels that provides information and actions that drive the business process that a work area supports. The Local Area is the focus of the users work. The contents of it should contain all of the information and actions required to accomplish the task.

The Contextual Area is the collapsible region on the right-hand side of the page that is filled with a column of panels. It provides additional space above the fold of the page to present actions and information, based on the information and state of the local area, that can assist the user in the task.
Designing Your UI

Your web user interface will be designed using the Oracle Fusion GPS concepts and design patterns. Many of these designs are delivered through the Oracle Fusion Middleware Extensions for Applications (Applications Core). Dashboards and Work Areas are built using the UI Shell. A set of design-time wizards and components that help support many of the Oracle Fusion GPS design patterns is also provided. These components, with those provided by Oracle ADF and WebCenter, provide the basis for all web interfaces.

13.4 Introduction to the UI Shell

The UI Shell is a page template containing default information, such as a logo, menus and facets. To supplement the UIShell template, there also is a UIShellMainArea template. Because you can load information into dynamic tabs, the Main area cannot be a part of the page itself since it is loaded dynamically. The UIShellMainArea template helps you create the flows that run within the tabs.

The UI Shell design supports task-based and user-based navigation and way-finding, and organizes screen real estate more effectively by collating tasks, providing dedicated spaces for primary-task supporting information, and maintains general order and appropriate hierarchy between various elements on the screen.

The UI Shell for Applications User Experience (Applications UX) patterns provides a system of containers that fulfill common layout and navigational requirements in a structured, consistent manner. The UI Shell focuses on providing detailed design for defining and organizing various types of navigation and other functionality such as search and auxiliary information for Oracle Fusion alone.

In particular, the UI Shell template supports:

- Global Search
- Navigation menus
- Cross-application navigation

For more information, see Implementing the UI Shell.

13.5 Applications UI Patterns and Features

Applications UI Patterns are high-level UI composite components that encapsulate standards and guidelines for common layouts, behaviors and flows across Oracle Fusion Applications, as set forth by the Applications User Experience group. The objective is to provide applications development teams with a higher level starting point and reduce duplication of effort in building the UI for their applications, while adhering to Oracle Fusion standards. The standards and guidelines are tightly integrated with JDeveloper.

Patterns can be implemented as custom components, declarative components or task flows. Patterns that are implemented as declarative components wrap the mandatory and pattern-specific UI components within the declarative component.

The UI patterns components provide several key benefits for developers when they are building pages and fragments:

- Enforcement of patterns.
- Faster development.
• Changes can be made to one component definition rather than to each instance in every application.

Supported patterns are:

• Applications Tables
• Applications Panels
• Applications Master-Detail
• Applications Detail On Demand
• Applications Tree
• Applications Tree Tables
• Applications Dialog Details
• Using the Custom Wizard with Applications Popups

**Applications Tables**

Applications tables are UI components that already contain an ADF table, a menu bar, a toolbar, and related popups. Developers do not need to create and assemble all these components separately.

**Applications Panels**

Applications Panels help you create the following UI components as part of the UI applications patterns:

• Page title
• Form title
• Page button bar (including navigation bar)
• Facets for page-specific UI components

You must use Applications Panels to standardize layout and appearance for all your page forms and buttons, including read-only pages.

**Applications Master-Detail**

Master-Detail refers to the interaction of selecting an object from a master list, and refreshing the details in an adjacent area. It is not the relationship of the data.

The Master-Detail composite is used in situations where the information is too large, dynamic, or complex to show in a flat table. The user can see the Master, or summary, information in one area, and the corresponding details in a separate area. This can be achieved using different master and detail components, such as table, tree table, and tree.

For instance, when the user selects an employee from the master table, the corresponding employee details are displayed in the region below in a label/data format.

**Applications Detail On Demand**

Dialog details are appropriate for use when information must be accessed quickly and then dismissed. The details are shown in a modeless dialog window.

Dialog details are accessed by clicking a details icon in a row in a table.
Applications Tree
The Applications Tree component provides these basic capabilities:

- Tree toolbar with default buttons
- Facets for adding ADF tree, custom toolbar buttons, and so on
- Default implementations for tree actions

Applications Dialog Details
The Applications Dialog Details component provides a user interface for launching a popup that contains detail information. Popups are an option when editing rows. The UI can be a detail icon, a link, or a button.

Using the Custom Wizard with Applications Popups
af:popup is a generic function documented in the Oracle Fusion Middleware Developing Web User Interfaces with Oracle ADF Faces guide.

While the standard af:popup component does not provide buttons or data binding, the Applications Popup wizard provides the base af:popup with:

- a title
- standard buttons
- customized button capability
- data binding
- code that developers can use to invoke the popup
- design-time support
- popup facets and properties that can be customized

Popups can be used as standalone components or with certain patterns.

Applications Tree Tables
The Application Tree Table component provides these basic capabilities:

- Tree Table toolbar with default buttons
- Facets for adding items such as ADF tree table and custom toolbar buttons
- Default implementations for tree actions

For more information, see:

- Implementing UIs in JDeveloper with Application Tables_ Trees and Tree Tables
- Implementing Applications Panels_ Master-Detail_ Hover_ and Dialog Details
This chapter discusses the UI Shell page template used to build web pages, and the components used to implement user interface features in Oracle JDeveloper (JDeveloper), such as menus and task flows.

This chapter includes the following sections:

- Introduction to Implementing the UI Shell
- Populating a UI Shell
- Implementing Application Menu Security
- Controlling the State of Main and Regional Area Task Flows
- Working with the Global Menu Model
- Using the Settings and Actions Menu
  - How to Implement End User Preferences
  - How to Use the Administration Options
  - How to Use the Troubleshooting Options

For more information about the features, see:

- Getting Started with Your Web Interface
- Implementing Search Functions in the UI Shell
- Implementing Additional Functions in the UI Shell
- Implementing UIs in JDeveloper with Application Tables, Trees and Tree Tables
- Implementing Applications Panels, Master-Detail, Hover, and Dialog Details

### 14.1 Introduction to Implementing the UI Shell

The UI Shell is a page template containing default information, such as a logo, menus, and facets. To supplement the UI Shell template, there is a UIShellMainArea template. Because you can load information into dynamic tabs, the main area (the center and the right as shown in Figure 14-1) cannot be a part of the page itself because it is loaded dynamically.

The UI Shell design supports task-based and user-based navigation, and organizes screen real estate more effectively by collating tasks, providing dedicated spaces for primary-task supporting information, and maintains general order and appropriate hierarchy among various elements on the screen.
The UI Shell for Applications User Experience (Applications UX) patterns provides a system of containers that fulfill common layout and navigational requirements in a structured, consistent manner. The UI Shell focuses on providing detailed design for defining and organizing various types of navigation and other functionality such as search and auxiliary information for Oracle Fusion Middleware alone.

**Note:**

Two UI Shell templates are implemented: the classic shell and the Simplified Experience UI (usually called FUSE+). Items can be shown or hidden based on the template. The API `getShellType()` will return "simple" or "classic" depending on the page template used in the current page. Developers can determine the active template by using the Expression Language expression `#{applcoreShellBean.shellType}` to identify the shell type.

---

**Before You Begin:**

You should be familiar with JDeveloper, be able to create and run JavaServer Faces (JSF) pages, and be able to create an Oracle Application Development Framework (Oracle ADF) task flow.

### 14.1.1 Standard Related to the UI Shell

Almost all shipped Oracle Fusion Applications pages are built using the UIShell page template. Exceptions include the login page, and the password preferences page.

### 14.1.2 UI Shell Description

The UI Shell is composed of four default areas: global, regional, local, and contextual, as shown in Figure 14-1.
The basic information about the UI Shell includes:

- The shell is optimized for a screen resolution of 1280x1024 pixels.
- The four areas (refer to the circled numbers on the image) are:

  1. **Global Area**: The global area, across the full width at the top of the UI Shell, is static, consistent, and persistent for an individual user. It contains controls that generally affect the contents of the other three areas. See Global Area Standard Icons.

  2. **Regional Area**: The regional area is in the left-hand pane of the UI Shell. It has controls and content that generally affect the contents of the local and contextual areas. Tasks lists in the regional area automatically are bulleted to make it clear when a line item wraps to the next line.

  3. **Local Area**: The local area is in the center of the UI Shell where users do their work. It is the main work area and typically contains the transaction form with the menus and controls that enable users to be productive. Controls in, and the content or state of, the local area generally affect the contents of the contextual area.

     - **Main Area**: This term designates the combination of the local area and the contextual area.

  4. **Contextual Area**: The contextual area is in the right-hand pane of the UI Shell, with controls and contents that generally are affected by controls in, or the content or state of, the local area; although in specific cases the contextual area can also affect the contents of the local area (causing a local-area reload).
• Application designers, customers, and administrators can set the regional and contextual areas as collapsed for specific applications. End users can expand those areas at runtime.

• If there is no content in the regional or contextual area, the area is collapsed and the ability to expand it is disabled.

• The contextual area is directly bound to the local area. The application developer can bind the contextual area content to the local area such that each invocation of a local area automatically causes a relevant contextual area in the correct state to appear alongside the local area.

14.1.2.1 Global Area Standard Icons

The global area incorporates several built-in icons, as shown here:

Figure 14-2  Global Area Standard Icons

From 1 (Home) to 10 (User Name), the numbered icons are:

• (1) Home

  Click this icon to return to the defined Home page. See Implementing the Oracle Fusion Home Page UI.

• (2) Navigator

  The Navigator, shown in Figure 14-23, is rendered when the Navigator icon is clicked on the UI Shell. See Displaying the Navigator.

• (3) Favorites, Recent Items and Tags

  This icon combines the Favorites, Recent Items and Tags functions. When you click this icon, the dialog shown in Figure 14-3 is displayed.

  Figure 14-3  Favorites, Recent Items and Tags

  – Favorites adds the currently-focused task flow to the Favorites list.
  – Recent Items tracks a list of the last 20 task flows visited by a user. See Implementing Recent Items.
  – Tags is a service that allows users to add tags to selected resources in Oracle Fusion Applications to contribute to resources other users have visited. See Implementing Tagging Integration.

• (4) Watchlist
Click the Watchlist icon to display a user-accessible UI that provides a summary of items the user can track using shortcuts. See Implementing the Watchlist.

- (5) **Notification**
  Click the icon to display the Notification dialog shown in Figure 14-4.

  **Figure 14-4  Recent Notifications Dialog**

<table>
<thead>
<tr>
<th>Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recent Notifications</td>
</tr>
<tr>
<td>🌐 Action Required: Saved Transaction</td>
</tr>
<tr>
<td>1/29/13 05:27</td>
</tr>
<tr>
<td>More... (0 unread)</td>
</tr>
</tbody>
</table>

- (6) **Social**
  Click the Social icon to display Oracle Social Network conversations. See Implementing Oracle Social Network Integration.

  **Note:**
  - The link displays only if Oracle Social Network is provisioned on the system and registered in the application topology.
  - The link is supplied automatically by the UI Shell; it requires no developer work.

- (7) **Accessibility**
  The Accessibility icon appears on all pages. It will allow users to set their accessibility preferences because the Personalization menu, which includes preferences, is hidden for anonymous users.

- (8) **Search**
  Click the icon to display the Global Search dialog shown in Figure 14-5.

  **Figure 14-5  Global Search Dialog**

<table>
<thead>
<tr>
<th>Close</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
</tr>
</tbody>
</table>

  See How to Use Basic Search.

- (9) **Help**
  When you click the Help icon, it turns orange. Then, if there is contextual help on the page, smaller orange icons will display, as shown in this figure.
14.2 Populating a UI Shell

The UI Shell is a page template with some facets for content that may be placed directly on the page, but it usually has its content inserted dynamically. The dynamic insertion happens by reading metadata in the form of menu metadata. This informs the UI Shell about which task flows to load and where. The UI Shell also can create a list of tasks, from the same metadata, that, when clicked, can load into the main area. All task flows and the page built by the UI Shell template follow the standard ADF Security framework.

When you create an application using the Oracle Fusion Web Application (ADF) template, two projects automatically are created for you: the model and the user interface projects. The default names for these projects that JDeveloper provides are Model and ViewController. You then add the Applications Core (ViewController) tag library to the user interface project. See Adding the Applications Core Tag Library to Your User Interface Project.

Related Links
The following document provides additional information related to subjects discussed in this section:
• For more security framework information, see the Oracle Fusion Middleware Developing Fusion Web Applications with Oracle Application Development Framework guide.

14.2.1 How to Create a JSF Page

Creating an Oracle Fusion application also creates your application’s workspace, where you will later place your page fragments and task flows. For more information about task flows, see the “Getting Started with ADF Task Flows” section of the Developing Fusion Web Applications with Oracle Application Development Framework.

For more information about starting JDeveloper, see Chapter 2, “Setting Up Your Development Environment.”

To create a JSF page:

1. Select the user interface project in the Application Navigator.

2. Choose File > New > Web Tier > JSF > JSF Page.

The Create JSF Page dialog is displayed, as shown in Figure 14-8.

3. In the dialog:

   • Enter a file name and directory path. The file name should follow the naming patterns required by your group.

   • From the Use Page Template list, select UIShell.

   • Select Create as XML Document (*.jspx).

   • Click OK.

4. The new JSF page is displayed in the editor. Note that the page headers in the Design view do not display exactly as they will at runtime.
**Note:**

This JSPX page is just the container for the UI Shell template. All other page content, such as the regional, local, and contextual area flows, and the dynamic task flows, are defined independently. At runtime, the menu definition assembles the various parts. All task flows are loaded into a page created with the UI Shell template by configuring the Menu file. This is done to control the behavior and the dynamic loading of task flows at runtime. It also creates the Navigator and Task List menu. See *Working with the Global Menu Model* and *Working with the Applications Menu Model*.

Now you can add components to the page. Table 14-1 lists the `itemNode` properties that can be used for a JSF page. See *Working with the Applications Menu Model* for how to add a menu to the page.

**Table 14-1  itemNode Properties of a JSF Page**

<table>
<thead>
<tr>
<th>ItemNode Property</th>
<th>Property Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>Name that has been assigned to the action.</td>
<td>Go to the page defined by the action.</td>
</tr>
<tr>
<td>dataControlScope</td>
<td>String</td>
<td>The values are shared (the default) or isolated. This is set at the page level <code>itemNode</code>. When <code>dataControlScope</code> is set to <code>isolated</code>, the UI Shell loads the main area and regional area task flows with <code>dataControlScope</code> set to <code>isolated</code>. When <code>dataControlScope</code> is set to <code>shared</code>, the UI Shell loads the main area and regional area task flows with <code>dataControlScope</code> set to <code>shared</code>. For example: &lt;itemNode id=&quot;itemNode_AppsPanelTests_TabsWA&quot; label=&quot;label_AppsPanelTests_TabsWA&quot; action=&quot;AppsPanelTests_TabsWA&quot; focusViewId=&quot;/AppsPanelTests_TabsWA&quot; dataControlScope=&quot;isolated&quot;&gt;</td>
</tr>
<tr>
<td>ItemNode Property</td>
<td>Property Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>isDynamicTabNavigation</td>
<td>True or False</td>
<td>This property provides an option to suppress dynamic tab navigation and just display one main area at a time. To do this, add the following property and value to the itemNode that represents your JSPX: isDynamicTabNavigation=&quot;false&quot; Other menu metadata stays the same. The Task List menu will continue to render. Clicking a Task link will replace the current main area task flow with the new one. Multiple defaultMain definitions are allowed and will open multiple tabs on page load. The first one with disclosed=&quot;true&quot; will be the tab in focus. If the property value is not defined, it defaults to true.</td>
</tr>
</tbody>
</table>
| id                     | Unique identifier | This is what appears in the work area title.  
Note: For all UI Shell work area pages with Data Visualization Tool (DVT) components in the default main area task flow, and for Home pages with DVT components, you must create the af:document title as an Expression Language expression that sets the title with the default main area task flow label, as shown in this example: <af:document id="d1" title="#{adfBundle['oracle.apps...resource.xyzGenBundle']['Header.DefaultMain']} - #{adfBundle['oracle.apps...resource.xyzGenBundle']['Header.WorkAreaLabel']} - #{adfBundle['oracle.apps.common.acr.resource.ResourcesGenBundle']['Header.OracleApplications']}" > For UI Shell pages with DVT components in their dynamic main area task flows, the title is set on the AdfRichDocument by UI Shell code for the openMainTask, closeMainTask and tab switch. |
| label                  | String          | This is what appears in the work area title.  
Note: For all UI Shell work area pages with Data Visualization Tool (DVT) components in the default main area task flow, and for Home pages with DVT components, you must create the af:document title as an Expression Language expression that sets the title with the default main area task flow label, as shown in this example: <af:document id="d1" title="#{adfBundle['oracle.apps...resource.xyzGenBundle']['Header.DefaultMain']} - #{adfBundle['oracle.apps...resource.xyzGenBundle']['Header.WorkAreaLabel']} - #{adfBundle['oracle.apps.common.acr.resource.ResourcesGenBundle']['Header.OracleApplications']}" > For UI Shell pages with DVT components in their dynamic main area task flows, the title is set on the AdfRichDocument by UI Shell code for the openMainTask, closeMainTask and tab switch. |
| focusId                | Name of the view activity | This is what appears in the work area title.  
Note: For all UI Shell work area pages with Data Visualization Tool (DVT) components in the default main area task flow, and for Home pages with DVT components, you must create the af:document title as an Expression Language expression that sets the title with the default main area task flow label, as shown in this example: <af:document id="d1" title="#{adfBundle['oracle.apps...resource.xyzGenBundle']['Header.DefaultMain']} - #{adfBundle['oracle.apps...resource.xyzGenBundle']['Header.WorkAreaLabel']} - #{adfBundle['oracle.apps.common.acr.resource.ResourcesGenBundle']['Header.OracleApplications']}" > For UI Shell pages with DVT components in their dynamic main area task flows, the title is set on the AdfRichDocument by UI Shell code for the openMainTask, closeMainTask and tab switch. |
Table 14-1  (Cont.) ItemNode Properties of a JSF Page

<table>
<thead>
<tr>
<th>ItemNode Property</th>
<th>Property Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formUsesUpload</td>
<td>True or False (default)</td>
<td>To set the UI Shell's af:formUsesUpload value to &quot;true,&quot; add the formUsesUpload=&quot;true&quot; property to the itemNode that represents the JSPX (similar to the way isDynamicNavigation is set). You would use this property when you want to allow uploading of file features using the af:inputFile component.</td>
</tr>
<tr>
<td>regionalAreaWidth</td>
<td>Numeric value</td>
<td>See How to Control Regional Area Task Flows.</td>
</tr>
<tr>
<td>isRegionalAreaCollapsed</td>
<td>True or False</td>
<td>See How to Control Regional Area Task Flows.</td>
</tr>
</tbody>
</table>

14.2.1.1 Working with the Applications Menu Model

Page and task flow information are local to a particular Oracle Fusion application and are exposed using the Applications menu model.

An Applications menu is related to a local JSPX file and includes the task lists, defaultMain, and defaultRegional. A menu is created for each Java EE application.

14.2.1.1.1 How to Create an Applications Menu

The following information describes how to create an ADF menu to access page elements through the Navigator on JSF pages or task flows that are based on the UI Shell template.

Select the JSPX page in the Application Navigator, then right-click and select the Create Applications Menu option.

This step creates the menu file with one itemNode. The menu file will be named `<view id>_taskmenu.xml`. For example, if there is a PageA.jspx, its view ID in the adfc-config.xml file is PageA, and the menu file name is PageA_taskmenu.xml. This step also should add the ApplicationsMenuModel managed bean entry into the adfc-config.xml file. The managed bean entry should not have the topRootModel managed bean property set. Example 14-1 shows a sample of the generated content in the PageA_taskmenu.xml file.

Creating ADF Menus for Multiple JSF Pages

Use this alternate method to create menus for multiple pages at once. This will create an empty menu on each page.

1. Right-click the adfc-config.xml file and choose Open to display the file in the JDeveloper editor.

   The adfc-config.xml file is located in the following location: `<project_name> > WEB INF`.

2. Drag pages from the Application Navigator to the adfc-config.xml file in the editor.

   View nodes that represent the pages or task flows are displayed in the editor.
When you drag the pages or task flows, they automatically are grouped into the same menu.

3. Right-click the adfc-config.xml file and choose Create Applications Menus. An empty panel, as shown in Figure 14-9, is displayed.

![Figure 14-9 Initial Create Applications Menus Display](image)

4. Click Find Pages to populate the display with all the JSPX pages that have been added to the adfc_config.xml file. As shown in Figure 14-10, pages that do not yet have a menu associated with them will have a checkbox that defaults to being selected, and pages that already have an associated menu are shown with a checkmark to the right.

![Figure 14-10 Populated Create Applications Menus Display](image)

5. Click OK to automatically create a menu file for each selected page and to update the task flow with a managed bean.

**Note:**

Menu files will follow the `<focusViewId>_taskmenu.xml` naming standard. For example, the menu file for the Example.jspx file will be `/WEB-INF/menus/Example_taskmenu.xml`.

When you create this menu file, the following occurs:

- The menu file is generated and placed into the following directory: `ViewController/public_html/WEB-INF/menus`
- The following are generated and appear in the adfc-config.xml file:
  - A new control-flow rule
  - A managed bean entry

The new menu that contains your pages is now accessible from the Navigator panel.
Example 14-1   Example of Generated Content in a taskmenu.xml

<?xml version="1.0" encoding="UTF-8" ?>
<menu xmlns="http://myfaces.apache.org/trinidad/menu">
<!-- This is the page level node -->
<itemNode id="itemNode_PageA" label="label_PageA" action="adfMenu_PageA"
focusViewId="/PageA"/>
<!-- Optional itemNode for Regional Task List task flow. Your task menu
definition may omit this if your page does not display a Regional Task List. -->
<itemNode id="__myProduct_RegionalTaskList"
focusViewId="/PageA"
label="#{applcoreBundle.TASKS}" taskType="defaultRegional"
taskFlowId="/WEB-INF/oracle/apps/fnd/applcore/patterns/uishell/ui/publicFlow/TaskList.xml#TasksList"
disclosed="true"
parametersList="fndPageParams=#{pageFlowScope.fndPageParams}"/>
<!-- Typical itemNode entry for a task flow -->
<itemNode id="__myProductTFId"
focusViewId="/PageA"
label="#{myBundle.myProductTFxyz}" taskType="defaultMain"
taskFlowId="<fully qualified-TFid>"
disclosed="true"/>
<!-- .. Additional itemNodes for other task flows on PageA -->
</itemNode>
</menu>

14.2.2 How to Add Default Main Area Task Flows to a Page

When a user opens an application, he or she expects something to be displayed automatically. A task flow in the main area accomplishes this.

To add a task flow to the default Main Area:

   The Create JSF Page Fragment dialog shown here is displayed.

   Figure 14-11   Create New JSF Page Fragment Dialog

   ![Create New JSF Page Fragment Dialog]

   2. In the Create New JSF Page Fragment dialog:
      a. Enter a page fragment name, for example:
The file name should follow these patterns:

- `<Object><Function>.jsff`, such as PoSearch.jsff.
- `<Object>.jsff`, such as PurchaseOrder.jsff.

The page name should convey the object it presents (an employee, a supplier, an item, a purchase order, an applicant, and so on), and the function being performed (search, promote, hire, approve, view). For some pages, the object is sufficient.

For update or create pages, just the object is necessary (unless the create and update pages are different as shown in the examples).

Never give pages step number names, such as PoCreateStep1.jsff or PoCreateStep2.jsff. Always describe the page function, such as PoDesc.jsff or PoLines.jsff.

b. From the Use Page Template list, select UIShellMainArea.

---

**Note:**

The UIShellMainArea template is only for main area task flows, not regional area task flows.

---

3. Click OK.

The Local and Contextual area facets of the page fragment appear in the editor.

4. Choose File > New > JSF > ADF Task Flow to create a default task flow.

---

**Note:**

Most applications will have multiple task flows for the Regional, Local and Contextual areas. For instance, Figure 14-1 shows 12 task flows.

---

The Create Task Flow dialog shown here is displayed.
In the Create Task Flow dialog:

a. The file name defaults to task-flow-definition.xml. Enter a descriptive file name.

The file name should follow these patterns:

- `<Object><Function>.xml`, such as PoSearch.xml.
- `<Object>.xml`, such as PurchaseOrder.xml.

The page name should convey the object it presents (an employee, a supplier, an item, a purchase order, an applicant, and so on), and the function being performed (search, promote, hire, approve, view). For some pages, the object is sufficient.

For update or create pages, just the object is necessary (unless the create and update pages are different as shown in the examples).

Never give pages step number names, such as PoCreateStep1.jsff or PoCreateStep2.jsff. Always describe the page function, such as PoDesc.jsff or PoLines.jsff.

b. From the Use Page Template list, select UIShellMainArea.
5. Ensure that the JSF page fragment file is selected in the Application Navigator and is displayed in the Edit view.

   a. In the Edit view, click the Source tab.

   b. Locate the line `<f:facet name="localArea"/>`.

   c. In the Application Navigator, select an applicable task flow, such as the one you created in Step 4, to add to the `localArea`. This should be an XML file located under `ViewController > Web Content > WEB-INF > oracle > apps > application_name > ui > flow`.

   d. Drag and drop the appropriate flow from the Navigator pane to immediately following `<f:facet name="localArea"/>`.

   e. From the Create menu that is displayed, select Region.

      The `af:region` tag is nested within the `localArea` facet and code resembling that shown in Example 14-2 will be inserted after it.

      In the Structure window, an `af:region` entry is added following the `f:facet - localArea` entry.

   f. Note: The instructions for the remainder of step 5 are optional. If you do not need a `contextualArea`, go to Step 6.

      In the Structure window, select `f:facet - contextualArea`.

   g. In the Component Palette, select ADF Faces > Layout.

   h. Click Panel Accordion. An `af:showDetailItem` entry is created automatically under `af:panelAccordion`.

   i. In the Source view, find and highlight the new `af:showDetailItem ...>` entry.

   j. In the Application Navigator, select an applicable task flow, such as the one you created in Step 4, and drag and drop it onto the highlighted entry in the Source view.

   k. From the Create menu that is displayed, select Region.

   l. Click OK on the Edit Task Flow Binding dialog that is displayed.

      Code resembling that shown in Example 14-3 will be inserted after `af:showDetailItem ...>`.

6. Now that you have created the main area page fragment, you must wrap it in an ADF task flow.

   Again, select File > New > Web Tier > JSF > ADF Task Flow and in the Create ADF Task Flow dialog:

   a. Enter a descriptive name for the task flow.

      For example, enter `def_main_task-flow-definition.xml`. 

---

Note:

The `UIShellMainArea` template is only for main area task flows, not regional area task flows.
b. Ensure that the **Create as Bounded Task Flow** and the **Create With Page Fragments** boxes are selected.

Do not change the other default settings.

7. Click **OK**.

The new task flow is displayed as a blank visual editor in the JDeveloper middle section.

8. In the Application Navigator, select your recently created page fragment (.jsff file), and drag and drop it onto the editor.

The page fragment **itemNode** appears in the editor, as shown in **Figure 14-13**.

![Figure 14-13 Page Fragment itemNode](image)

9. To load the menu metadata:

---

**Note:**

The menu data accomplishes several important jobs for you:

- It defines properties of the page for you. For instance, it will be displayed in no-tab mode or with dynamic tabs, and define the width of the regional area.
- It can create a task list menu for each page.
- It can create labels for groups of tasks.

---

a. In the Application Navigator, select the **test_menu_taskmenu.xml** file that you created using the ADF Menu Model dialog. For details about creating the menu, see **How to Create an Applications Menu**.

b. In the **test_menu_taskmenu.xml** structure view menu tree, shown in **Figure 14-14**, right-click the **itemNode** item and choose **Insert inside itemNode <task_flow_name> > itemNode**.
The Insert itemNode - Common Properties dialog, shown in Figure 14-15, is displayed.

Figure 14-15  Insert itemNode - Common Properties Dialog

10. In the id field, enter a unique ID using this standard format:
    
    <pageID>_<taskFlowName>
    
    This ID example consists of the concatenated page ID (or page name), an underscore, and the task flow name, for example, ExpenseWorkArea_CreateExpense.

11. In the focusViewId field, enter the ADFc View Activity ID of the page under which you are registering the task flow.

12. Click Finish.
    
The new item node is displayed in the structure view, under the menu tree.

13. With the item node selected in the structure view, click the Property Inspector tab, as shown in Figure 14-16.
14. In the **label** field, enter a label for the task flow, such as **CreateExpense**.
   This label will be the title of the tab that is opened by the Task Type defaultMain.

   **Note:**
   Do not leave this field null. This is the label that will appear in the tab header when in a tabs page. Even if you are in a no-tabs page (see Supporting No-Tab Work Areas), do not leave it blank because this label will be used in other ways, such as Add to Favorites, or when the system tracks the Recent Items.

15. Select **test_menu_taskmenu.xml** in the Project Navigator tree, and your task flow in the structure view to display its Property Inspector, or select the Property Inspector tab.

   In the Advanced section of the Property Inspector, enter the following values:

   - **Task Type**: defaultMain (the task flow is displayed by default whenever the page is rendered).
     The Data Control Scope should have been set to **isolated**, inside the task flow definition for any task flow in the menu (defaultMain or dynamicMain) or any call from openMainTask. See **dataControlScope** in Table 14-1.

   - **Task Flow Id**: ID of the task flow to be loaded.
     To enter the ID, click the ellipsis to display the Select Task Flow Id dialog, shown in **Figure 14-17**, and browse to the task-flow definition location. By default, following the standard naming structure, the location will be in...
Click Open to automatically enter the location in the Task Flow field. The task flow ID is a concatenation of the file location for the task-flow definition and the task-flow name. It typically resembles /WEB-INF/MyTaskFlow.xml#MyTaskFlow. The Property Inspector for the itemNode should resemble the example shown in Figure 14-16.

16. To run the JSPX page, select the page in the Application Navigator, right-click the page file, and choose Run.

The new page, similar to that shown in Figure 14-18, is displayed in a web browser.

17. Check that the newly rendered page contains one tab whose content is the task flow that you defined in this procedure.
The available `itemNode` properties for main area and regional area task flows for application menus are shown in Table 14-2.

<table>
<thead>
<tr>
<th>ItemNode Property</th>
<th>Property Value</th>
<th>What Happens on the Rendered Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>taskType</code></td>
<td>Note: <code>taskType</code> can have four values:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• dynamicMain</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• defaultMain</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• defaultRegional</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• taskCategory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If the value is <code>dynamicMain</code>, the page contains a new link in the regional area. When you click the link, a new tab with the loaded task opens. If the no-tabs model is used, no new tab is opened. Rather, the current main area contents are replaced. (See Supporting No-Tab Work Areas.)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If the value is <code>defaultMain</code>, the page contains a tab already running this task in the main area. If the no-tabs model is used, only one <code>itemNode</code> should be defined as a <code>defaultMain</code>. (See Supporting No-Tab Work Areas.)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If the value is <code>defaultRegional</code>, the task is loaded into the regional area.</td>
<td></td>
</tr>
<tr>
<td><code>label</code></td>
<td>String</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Note: When passing parameters, do not leave the label field null. This is the label that would appear in the tab header when in a tabs page. Even if you are in a no-tabs page (see Supporting No-Tab Work Areas), do not leave it blank because this label will be used in other ways, such as Add to Favorites, or when the system tracks the Recent Items.</td>
<td></td>
</tr>
<tr>
<td><code>taskFlowId</code></td>
<td>ID of the task flow to be loaded. The task flow ID is a concatenation of the file location for the task flow definition, and the task flow name. For example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/WEB-INF/MyTaskFlow.xml#MyTaskFlow</td>
<td></td>
</tr>
<tr>
<td><code>reuseInstance</code></td>
<td>True or False</td>
<td></td>
</tr>
</tbody>
</table>
| (optional)        | If `True`, when the link is clicked a second time, the tab is brought to the top. A `False` value means that clicking the corresponding task link opens new tabs. However, if the no-tabs model is used, no new tab is opened. Rather, the current main area contents are replaced. (See Supporting No-Tab Work Areas.)
Table 14-2  (Cont.) itemNode Properties for Main and Regional Task Flows for Application Menus

<table>
<thead>
<tr>
<th>itemNode Property</th>
<th>Property Value</th>
<th>What Happens on the Rendered Page</th>
</tr>
</thead>
</table>
| keyList           | String         | Important: keyList is used with the task flow ID to identify the target tab in the main area. As such, keyList is only applicable in dynamic tabs mode, and is ignored in no-tabs mode. keyList provides a way to identify a task flow instance. When reuseInstance is true, use the specified keyList in addition to the task flow ID to identify the target tab. The keyList parameter has been implemented for the following FndUIShellController data control methods:  
  • openMainTask  
  • discloseRegionalTask  
  • collapseRegionalTask  
  • navigate  
  • openSubTask  

In dynamic tabs mode, when looking for a match of an existing tab, these APIs will first look for any instances of the task flow that is already open, which has the same task flow ID as the one passed into them as the parameter. In addition, it will compare the keyList values, such that the existing task flow will be picked only if its keyList values match the ones specified in the keyList parameter. It does not matter if the task flow parameters are the same or different. If the keyList is not set in the menu metadata, you can reuse a tab only if you pass in a null keyList. |
<p>| loadPopup         | True or False  | See Implementing the Task Popup. This provides a way to load the task flow into a Popup when the user clicks one of the Task List links. |</p>
<table>
<thead>
<tr>
<th>itemNode Property</th>
<th>Property Value</th>
<th>What Happens on the Rendered Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>loadDependentFlow</td>
<td>True or False</td>
<td>The no-tab navigation mode can load a main area task flow and a dependent task flow simultaneously, while displaying only one flow at a time. (See Supporting No-Tab Work Areas.) The UI Shell is limited to 12 task flows: 10 tabs in tab mode; 1 tab in no-tab mode and 1 dependent in no-tab mode. Dependent Flow is applicable only to the no-tab navigation model. Instead of having only one region for the no-tab navigation model, there are two regions: one for the main task flow and another for the dependent flow. These regions are in a switcher, so that only one is visible at a time. If a dependent task flow is loaded, only the dependent task flow region is shown, and the main task flow region is hidden. When the dependent task flow is closed, the main task flow region is redisplayed, with its state preserved. When loadDependentFlow is true, the openMainTask API will load the target task flow in the dependent region. Loading a new task flow in the main task flow will close both the existing main task flow and, if any, the existing dependent task flow. Loading a new task flow in the dependent task flow will replace only the existing dependent task flow, if any, and leave the main task flow intact.</td>
</tr>
<tr>
<td>forceRefresh</td>
<td>True or False</td>
<td>If forceRefresh = true, the contents are refreshed. If forceRefresh is set to false, if the task flow parameters are identical, no refresh will occur, but if they are different, the task flow is refreshed using the new parameters.</td>
</tr>
<tr>
<td>stretch</td>
<td>True (default) or False</td>
<td>When the UIShellMainArea stretch attribute is set to true, contents under localArea will be stretched when rendered in the local area. When set to false, contents under localAreaScroll will not be stretched, but will be rendered with a scroll bar, if necessary, in the local area. (This facet is contained within an af:panelGroupLayout with layout=scroll.)</td>
</tr>
<tr>
<td>disclosed</td>
<td>True or False</td>
<td>All the task flows will be rendered in the main area. The task flow that has disclosed set to true will be in focus. More than one defaultRegional task can have a true disclosed value, because more than one detail item may be disclosed at a time under a panelAccordion component. If the disclosed value is true, the regional area is expanded. If the disclosed value is false, the regional area is collapsed.</td>
</tr>
</tbody>
</table>
### Table 14-2  (Cont.) itemNode Properties for Main and Regional Task Flows for Application Menus

<table>
<thead>
<tr>
<th>itemNode Property</th>
<th>Property Value</th>
<th>What Happens on the Rendered Page</th>
</tr>
</thead>
</table>
| active            | True or False (default) | Task flow definitions use conditional activation. There are several cases in which Oracle Fusion Applications run with the regional area collapsed by default. Unless the user expands it, there is no need to activate the task flows for the regional area. However, some use cases depend on the task flow that is under the regional area being active even when collapsed. In this case, the active attribute can be set in the property inspector for the item node. active has three possible values:  
  - default <False>  
  - False  
  - True  
  If you require that your task flows be activated or run even though they are not displayed, you must change the active property on the itemNode to True. |
| taskFlow          | String         | The Task List is exposed as a task flow. See *How to Pass Parameters into Task Flows from Tasks List*. |
| destination       | String         | The destination attribute is supported on the item nodes for Task List; that is, for item nodes that have the task type set to dynamicMain. The destination attribute is intended only for navigating to an external web site. When it is defined, it takes precedence over all other attributes. Example of the menu data:  
  <itemNode id="__ServiceRequest_itemNode_externalUrl" destination="http://www.yahoo.com"/> |
| parametersList    | String         | Use to specify which parameters to pass into a task flow. The value of this property is a delimited list of parameter name-value pairs. See *How to Pass Parameters into Task Flows from Tasks List*. |
| contextCode       | Domain name in which the J2EE application belongs. Valid values are: CommonDomain, BI, FIN, SCM, CRM, PRJ, HCM, PRC, or IC. | This value would be used for the Auto-suggest feature in Global Search. See *Implementing Auto-suggest*. |
| objectType        | String         | Classification for the Recent Item object (for example: opportunity, contact, lead, quote). This value would be used for the Auto-suggest feature in Global Search. |
Table 14-2  (Cont.) itemNode Properties for Main and Regional Task Flows for Application Menus

<table>
<thead>
<tr>
<th>itemNode Property</th>
<th>Property Value</th>
<th>What Happens on the Rendered Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectName</td>
<td>String</td>
<td>User recognizable string for a given Recent Item instance. This value is used as a string to match keystrokes entered in the auto-suggest field of Global Search. This value will be in the language of the user at the point of navigation (that is, when the record is saved). It will not change if the user changes language.</td>
</tr>
</tbody>
</table>

Example 14-2  Creating Region localArea Facet Added Code

```xml
<af:region value="#{bindings.EmpCreateUpdateFlow1.regionModel}"
    id="EmpCreateUpdateFlow1"/>
</f:facet>
```

Example 14-3  Example Edit Task Flow Binding Code

```xml
<af:region value="#{bindings.EmpSummaryTF2.regionModel}"
    id="EmpSummaryTF2"/>
```

14.2.3 How to Add Dynamic Main Area and Regional Area Task Flows to a Page

Unless otherwise noted, follow the procedure outlined in How to Add Default Main Area Task Flows to a Page, to insert the appropriate itemNode properties listed in Table 14-2.

- To add a dynamic main area task flow, set `taskType="dynamicMain"`.
- To add a default regional area task flow, set `taskType="defaultRegional"`.

14.2.3.1 Adding the Tasks List Menu to the Page

A tasks list is not a default widget as part of the UI Shell regional area. A tasks list is packaged as an ADF Controller task flow. You must manually add this task flow as you would any other defaultRegional task.

Specify the tasks list task flow as a defaultRegional task explicitly. If you do not do this, the tasks list does not render.

Add the following entry to your `menu.xml` file before the item node of tree structure and tree versions:

```xml
<itemNode id="__YourPage_itemNode__FndTasksList"  
    focusViewId="/YourPage" label="#{applcoreBundle.TASKS}" 
    taskType="defaultRegional"
    taskFlowId="/WEB-INF/oracle/apps/fnd/applcore/patterns/ 
    uishell/ui/publicFlow/TasksList.xml#TasksList"
    disclosed="true" 
    parametersList="fndPageParams=" 
        #{pageFlowScope.fndPageParams}"/>
```

**Note:** The `taskFlowId` value path must appear in a single line to avoid an exception during runtime.

- `Id` must be unique within the menu metadata.
- focusViewId is the focusViewId of your page.
- Set label to the default label provided by the Oracle Fusion Middleware Extensions for Applications (Applications Core).
- taskType should be defaultRegional.
- taskFlowId should point to the tasks list task flow provided by Applications Core.
- The disclosed attribute is usually set to true. Although, it can be set to false if you do not want to disclose tasks list by default.
- parametersList should set fndPageParams as previously shown so that this object is available in the pageFlowScope of the tasks list task flow. This context is necessary for Single Object WorkArea. For more information, see Using the Single Object Context Workarea.

14.2.3.2 Group Tasks in the Tasks Pane into a Category

The Task Category is a label that is used to group tasks in a task list.

1. In the Structure window for the menu, right-click the page itemNode whose taskType value is dynamicMain and choose Surround with...

   The Surround dialog is displayed.

2. Select itemNode and click OK.

   The Insert Item Node dialog page opens to the Common properties tab.

3. In the Common properties tab, enter these values:
   - id field: Concatenation of the page ID and a short category name, using the following format, is suggested:
     pageId_categoryName
     For example, you might enter: ExpenseWorkArea_NewExpense.
   - focusViewId: Click the ellipsis to open the Advanced Editor.
     Select the focusViewId of the page.

4. Click Finish.

5. In the Property Inspector for itemNode - ExpenseWorkArea_NewExpense, enter these values:
   - label: Name of the label, such as NewExpense.
     Do not leave this field blank. This is the label that would appear in the tab header when in a tabs page. Even if you are in a no-tabs page (see Supporting No-Tab Work Areas), do not leave it blank because this label will be used in other ways, such as Add to Favorites or when the system tracks the Recent Items.
Note:
The label should be defined in a resource bundle so it can be translated more readily.

- **Task Type:** taskCategory

6. Run the page by right-clicking the JSPX page file in the Projects tree view and choosing Run

   Confirm that the page contains task links arranged by category. The Task List is in the left regional area of the page. Items in the Task List are bulleted to make it clear when a line wraps.

### 14.2.3.3 Linking to a Task Flow in a Different Page

The Tasks Pane can link to a task flow in a different JSPX and pass page-level and task-level parameters.

The `navigateViewId` attribute supports this feature.

**Example 14-4** shows a sample of the metadata for a link in a Task List that links to another page.

**Example 14-4  Example Metadata for a Link in a Task List that Links to Another Page**

```xml
<ItemNode id="__ServiceRequest_itemNode__toTestPage1" focusViewId="/ServiceRequest" label="Go to different page" navigateViewId="/TestPage1" taskType="dynamicMain" taskFlowId="/oracle/apps/fnd/appcore/patterns/demo/SRTree.xml#SRTree"/>
```

### 14.2.3.4 Support No-Tab Work Areas

You can suppress dynamic tab navigation and just display one main area at a time. To do this, add `isDynamicTabNavigation="false"` to the `itemNode` that represents your JSPX page, as shown in **Example 14-5**.

Note that the default value of `isDynamicTabNavigation` is `true`.

You also can set the no-tab mode declaratively in the Property Inspector:

1. Select the `itemNode` from the Structure window.
2. Go to Property Inspector.
4. Set the Dynamic Tab Navigation property to `false`.

**Example 14-5  Implementing a No-Tab Work Area**

```xml
<ItemNode focusViewId="/SelTestWorkarea" id="stp1" taskType="dynamicMain" taskFlowId="/WEBINF/oracle/.../ProductMainFlow.xml#ProductMainFlow" label="#{adfBundle['oracle.apps....SelTestWorkarea_taskmenuBundle'].DEFINE_PRODUCT}" isDynamicTabNavigation="false"/>
```

Other menu metadata stay the same. Tasks List will continue to render. Clicking a Tasks Link will replace the current main area task flow with the new one.
14.2.3.5 Implementing the Task Popup

The Task Popup provides a way for you to:

- Load a task flow into a popup when the user clicks one of the Tasks List links.
- Cancel from the popup or open a new main area task flow, passing in parameter values from the popup.

Implementation Notes

The UI Shell provides an af:popup component with a modal af:panelWindow as its immediate child, which would contain a dynamic region defined in it. When selected by the user, the UI Shell will load the task flow into the dynamic region, and show the modal af:popup panelWindow without any buttons. Therefore, the task flow must include the OK and Cancel buttons that are used to open a dynamic tab and dismiss the popup, respectively. The dialog title will be set according to the label mentioned in the menu metadata of the dynamic task link. There is a refresh condition set on the dynamic region that refreshes the task flow and reloads it each time the popup is opened.

Remember the following when you implement the Task Popup feature:

- For a dynamicMain task that you would like to load into the popup, specify the loadPopup property as true. For example, as shown in Example 14-6, the ChooseSR task flow would be loaded in a popup when the user clicks its link in the Tasks List. The label that is mentioned will be displayed as the dialog title of the popup that starts the task flow.

- You can define any components within this task flow, except the af:popup and its child components, such as af:dialog, af:panelWindow, and af:menu.

- You cannot have a UIShellMainArea page template or any other templates inside the popup task flow.

- You must add the necessary buttons as part of the task flow. For example, if the task flow has a simple .jsff file, it should contain OK and Cancel buttons, along with other components.

- Create a managed bean to set the action listener for the Cancel button. See Implementing OK and Cancel Buttons in a Popup.

- Create another method for the OK button that calls the method in Example 18-5, and any additional processing logic. The common use case would be opening a new task in the main area by using the openMainTask API. For example, you can bind the OK button to a managed bean and add your own action listeners. See Implementing OK and Cancel Buttons in a Popup.

- You then can pass the parameters directly from the managed bean to the openMainTask API bindings for the popup task flow page to open a new dynamic tab. The menu data entries for parameters will not have any bearing on the dynamic taskFlow tab that they are loading in the main area. The details of that task flow should come from the openMainTask API that is bound to the OK button.

Example 14-6   Example Use of loadPopup Property


14.2.4 How to Pass Parameters into Task Flows from Tasks List

Item nodes with a `taskType` of `dynamicMain`, `defaultMain`, and `defaultRegional` have parameter support. In addition to specifying the `taskFlowId` to load when the user clicks a task flow link, you can specify which parameters to pass into that task flow. This is accomplished with the `parametersList` and `methodParameters` properties on the `itemNode`.

For the `itemNode` where you would like to specify parameter passing, add the `parametersList` property. The value of this property is a delimited list of parameter name-value pairs that will resemble Example 14-7.

Code in the managed bean for passing a hashmap to the task flow would resemble Example 14-8. A hashmap is a data structure that uses a hash function to map identifying values, known as keys (such as a person’s name), to their associated values (such as their telephone number).

Then, in the managed bean of the task flow, the Java object can be read, as shown in Example 14-9.

The `testValue` parameter is bound to an `af:outputText` value attribute, such as `<af:outputText value="#{TestPanelSplitter1MBean.testValue}"/>`, in the page fragment of the task flow.

**Example 14-7 Using the parametersList Property**

```xml
<itemNode id="__ServiceRequest_itemNode__SRDefault" focusViewId="/ServiceRequest"
  label="Pending Service Requests"
  taskType="dynamicMain"
  taskFlowId="/oracle/apps/fnd/appcore/patterns/demo/SRTable.xml#SRTable"
  parametersList="param1=value1;param2=value2;param3=#{ELForValue3}"/>
```

**Example 14-8 Example Code for Passing a Hashmap**

```java
private FndMethodParameters fndMethodParams;
...
public void setRichCommandLink1(RichCommandLink richCommandLink1)
{
    this.richCommandLink1 = richCommandLink1;
    FndMethodParameters methodParams = new FndMethodParameters();
    HashMap testHashMap = new HashMap();
    testHashMap.put("param1", "12345");
    testHashMap.put("param2", "67890");
    methodParams.setCustomObject(testHashMap);
    fndMethodParams = methodParams;
}
```

**Example 14-9 Reading Java Object in Managed Bean**

```java
public String getTestValue()
{
    Map pageFlowScope = 
        AdfFacesContext.getCurrentInstance().getPageFlowScope();
    Object custom = pageFlowScope.get("fndCustomObject");
    String outputTextString = "";
    if (custom != null && custom instanceof HashMap)
    {
        HashMap myHashMap = (HashMap)custom;
```
String temp1 = (String)myHashMap.get("param1");
String temp2 = (String)myHashMap.get("param2");
outputTextString = temp1 + temp2;
}
testValue = outputTextString;
return testValue;
}

### 14.2.5 How to Open Data Files from a Tasks List Link

The UI Shell implements this feature by using the **URLView** activity that is a task flow component of ADF. **URLView** generally is used to redirect the current request state of the application to an external or internal URL. With UI Shell, this **URLView** activity is only being used to open files that are internal to the current web application.

Therefore, the only input that the task list link will need is the internal path (within the webApp) of the file. After the path is provided to UI Shell, it will determine the current contextual root of the application and append it to the internal path of the file. After the Universal Resource Identifier (URI) for the file is generated, this is set on the **URLView** activity and an action expression is set on the task link to open the URL view. UI Shell also must call an **actionEvent** JavaScript method on the client side that will not allow the page to lose its current state upon redirection from the **URLView** activity.

**Note:**
You can only open data files that are part of your webApp, such as `/oracle/apps/fin/acc/file1.xls`. This feature supports only opening data files through the task list. Any other URI paths, such as a JSPX or a JSF page, are not supported.

---

**Implementation Notes**

For a **dynamicMain** task item that you would like to use to open the data file, there is a property called **filePath** in the Menu panel for the UI Shell page XML file. To enable this property in the Menu panel, during design time, the task type of the **itemNode** must be **dynamicMain**. The file URI path then should be specified against the **filePath** attribute in the Menu panel, as shown in Example 14-10.

You can specify any file type, such as xls, doc, pdf, txt, rtf, and ppt, that is within the application.

**Example 14-10  Specifying the File URI Path**

```xml
<itemNode id="__ServiceRequest_itemNode__ChooseSR"
    focusViewId="/ServiceRequest" label="Download File"
    taskType="dynamicMain" filePath="/WEB-INF/oracle/apps/Accounts.xls" />
```

### 14.3 Implementing Application Menu Security

Security of menus has two parts: actual access to the page or task flow, and the rendering of the menu itself. Any page or task flow is protected to run for a user only if that user has access to run the page or task flow. Directions for setting this up are in the Adding Security to an Oracle Fusion Web Application section in the *Developing Fusion Web Applications with Oracle Application Development Framework*.

Application menus and task list menus will automatically have their page security checked by the menu utilities. If the user does not have access, the menu entry will not
be rendered. If the following three conditions are true, security checks if a logged-in user has view privilege for a given task flow:

- The application has enabled authorization.
- The `taskType` is `dynamicMain` for the `itemNode`.
- The `taskFlowId` attribute is defined in the `itemNode`.

If any one of these conditions is not true, security is not checked and the `itemNode` will be protected only by the rendered attribute.

Application menus can have a security Expression Language expression on the rendered attribute that, if it returns false, will not render the menu entry. To do this, set the rendered attribute of the menu entry to an expression that evaluates to `true` or `false`. For instance, if the task list is to edit certain tax forms, this could be a business rule to hide or show links based on whether the customer is a nonprofit company. If it evaluates to false, the menu will not appear. For more information on all the security expressions, see the Adding Security to an Oracle Fusion Web Application section in the *Oracle Fusion Middleware Developing Fusion Web Applications with Oracle Application Development Framework*.

If your UI Shell pages are secured by ADF Security, you must add a policy similar to that in Example 14-11 to the `jazn-data.xml` file.

**Task Flow Example**

Bounded task flows are secure by default, and require the policy shown in Example 14-12.

If the policy is missing, then framework-level checks will prevent access to the task flow (typically by throwing an error).

But how would a menu item or command link disable or hide itself based on a preliminary check of the same permission? Example 14-13 shows the generic Expression Language expression being used to perform a preliminary check of the Task Flow Permission. Note that this is only needed for an `itemNode` with `taskType="defaultMain"` or "defaultRegional". The security check is performed automatically for an `itemNode` with `taskType="dynamicMain"` (that is, what is in the tasks list).

Example 14-14 shows the task flow-specific Expression Language expression.

Note that both of these checks actually go directly against the policy store; that is, they do not query the task flow definition. This avoids the overhead of loading a large number of ADF artifacts to render links and menus.

**Example 14-11 Adding a Security Policy to the jazn-data.xml File**

```xml
<grant>
  <grantee>
    <principals>
      <principal>
        <class>oracle.security.jps.internal.core.principals.JpsAnonymousRoleImpl</class>
        <name>anonymous-role</name>
      </principal>
    </principals>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.adf.share.security.authorization.RegionPermission</class>
    </permission>
  </permissions>
</grant>
```
14.4 Controlling the State of Main and Regional Area Task Flows

As described in How to Control Main Area Task Flows, UI Shell tasks to open or close a main area tab are exposed as data control methods so that you can create such UI artifacts through drag and drop. You do not need to create your own data control methods and manually raise Contextual Events.
Note:

openMainTask and closeMainTask only work when the calling task flow is loaded into the Main Area directly and locally.

For example, assume a UIShell page of Application1 that is running in Server1 is loading Taskflow1 in its Main Area. This Taskflow1 shows another task flow, Taskflow2, from Application2 running on Server2 and using a portlet mechanism or remote task flow mechanism.

If Taskflow2 uses openMainTask to open another task, it will not work. The task flow that invokes openMainTask should be loaded directly in the Main Area, and the task flow that is loaded should be available to the application locally.

14.4.1 How to Control Main Area Task Flows

The methodParameters parameter can be used to pass a Java object into the task flow that is specified in the taskFlowId parameter. Use the setCustomObject() method in FndMethodParameters to set the Java object. Note that passing a Java object is not supported in Navigate API.

Example of Passing a Java Object Using openMainTask

Bind the methodParameters parameter value to a managed bean property. Example 14-15 shows the methodAction binding in the page definition of the page fragment that calls openMainTask. Also see Table 14-2.

Data control APIs are:

- FndUIShellController.openMainTask

Note:

When passing parameters, do not leave the label field null. This is the label that would appear in the tab header when in a tabs page. Even if you are in a no-tabs page (see Supporting No-Tab Work Areas), do not leave it blank because this label will be used in other ways, such as Add to Favorites, or when the system tracks the Recent Items.

- FndUIShellController.closeMainTask. See closeMainTask History for more information.

To open or close a main area tab, drag and drop the appropriate data control method to create the UI functionality. Having specified the parameter values for these methods, user clicks will prompt the UI Shell to react accordingly.

To use the openMainTask data control method:

1. Expand the Data Controls and select the openMainTask item, as shown in Figure 14-19.
2. Drag **openMainTask** and drop it onto the page fragment. When you do, the Applications Context menu shown in Figure 14-20 is displayed so you can choose one of the three options.

![Figure 14-20](image)

To use the **closeMainTask** data control method:

1. Expand the Data Controls and select the **closeMainTask** item, as shown in Figure 14-21.

![Figure 14-21](image)

2. Drag **closeMainTask** and drop it onto the page fragment. When you do, the Applications Context menu shown in Figure 14-22 is displayed so you can choose one of the three options.
Bind the `methodParameters` parameter value to a managed bean property. Example 14-15 shows the `methodAction` binding in the page definition of the page fragment that calls the `openMainTask` method.

Example 14-8 shows code in a managed bean for passing a hashmap to the task flow.

Then, in the managed bean of the task flow, the Java object can be read, as was shown in Example 14-9.

The `testValue` parameter is bound to an `af:outputText` value attribute, such as `<af:outputText value="#{TestPanelSplitter1MBean.testValue}"/>`, in the page fragment of the task flow.

**Example 14-15  methodAction Binding in Page Definition of Page Fragment That Calls openMainTask**

```xml
<methodAction id="openMainTask" RequiresUpdateModel="true"
Action="invokeMethod" MethodName="openMainTask"
IsViewObjectMethod="false" DataControl="FndUIShellController"
InstanceName="FndUIShellController.dataProvider"
ReturnName="FndUIShellController.methodResults.openMainTask_FndUIShellController_dataProvider_openMainTask_result">
    <NamedData NDName="taskFlowId" NDValue="/WEB-INF/TestPanelSplitterTaskFlow#TestPanelSplitterTaskFlow" NDType="java.lang.String"/>
    <NamedData NDName="keyList" NDValue="" NDType="java.lang.String"/>
    <NamedData NDName="parametersList" NDValue="" NDType="java.lang.String"/>
    <NamedData NDName="label" NDValue="Test App Panel" NDType="java.lang.String"/>
    <NamedData NDName="reuseInstance" NDValue="" NDType="java.lang.Boolean"/>
    <NamedData NDName="forceRefresh" NDValue="" NDType="java.lang.Boolean"/>
    <NamedData NDName="loadDependentFlow" NDValue="" NDType="java.lang.Boolean"/>
    <NamedData NDName="methodParameters" NDValue="" NDType="oracle.apps.fnd.applcore.patterns.uishell.ui.bean.FndMethodParameters"/>
</methodAction>
```

14.4.1.1 closeMainTask History

The dynamic tabs mode tracks the last tab that was displayed before the current tab. When the current tab is closed, that last tab is brought back into focus.

In no-tabs mode, a stack of all the task flows that were opened is maintained, along with the parameter values. When the current task is closed, the task flow (with its original parameters) that was open before the current one, is reinitialized.

There are two ways in which the previous tab information is set for a given tab. When a new tab is opened, the tab that was in focus is the new tab's previous tab. When the user clicks a tab UI, the last tab that had the focus becomes the current tab's previous tab.

**Figure 14-22  Selecting a Close Option from the Applications Context Menu**
The `MainAreaHandler.handleOpenMainTaskEvent` method has a mechanism to handle the new tab. The managed bean for the tab adds an additional property for the previous tab. When a new tab is configured to be opened, the current tab is set as the previous tab for the managed bean for the new tab.

A disclosure listener, the `MainAreaBackingBean.setLastDisclosedItem`, handles user clicks in the tab UI. When the user clicks a tab, two events occur: one for the tab that is going out of focus, and one for the tab that is coming into focus. First, during the out-of-focus event, the tab that is going out of focus is captured in the managed bean's instance variable. Then, during the in-focus event, that instance variable's value is set as the previous tab in the managed bean for the newly focused tab.

Through user clicks, it is possible to create a circular dependency in which TabA's previous tab is TabB, whose previous tab is TabA. In this case, when TabA is closed, TabB would come into focus. However, when TabB is consequently closed, TabA would have to be focused, but it has already been closed. This corner case is handled by moving the focus to the first tab in the main area.

**No-Tab Navigation**

To keep track of all task flows that have been opened, a Stack instance variable is introduced in the `MainAreaHandler` method. When a new task flow is opened, the task flow ID and its associated parameter values are added to the stack.

Having this information, the call to `closeMainTask` pops the stack to get the last task flow ID and its parameter values that were displayed, and reinitializes the main area with that task flow and parameter information.

See also Supporting No-Tab Work Areas.

### 14.4.2 How to Control Regional Area Task Flows

The UI Shell exposes the means to control the disclosure state of the regional area as a whole, and the disclosure state of individual panels within the regional area panel accordian.

**Declarative support:** Allows you to specify the initial state of the following on loading a work area JSPX page.

- Within the regional area, whether a regional area task panel is collapsed or disclosed.
- A given regional area panel that is disclosed on initial rendering of the page should reflect its assigned pixel height to determine how much screen real estate it occupies.

**Programmatic support:** Allows you to control the initial or subsequent state of the following within a work area JSPX page.

- By default, the disclosure state is driven by what is specified declaratively. However, after initial page load, you can override the declarative default and, for example, render the work area with the regional area collapsed (overriding the declarative setting of rendering that work area with the regional area disclosed). Disclosing a collapsed regional area splitter programmatically in response to a UI action by the user (such as a button click or menu selection).

Declarative support is provided using attributes exposed on the respective item node in the Menu Model.

For regional panels:
There are separate APIs that expose parameters to refresh the task flow and set the disclosure state for the showDetail items in the panel accordian. The showDetail items are identified by the task flow ID specified.

**Implementation Notes**

- Specify the default values for the regional or main splitter position and collapsed state in the menu for the item node that represents the page, using the `regionalAreaWidth` and `isRegionalAreaCollapsed` properties. A sample entry in the menu file resembles Example 14-16.

- If these properties are not set in the menu for the top-level item node that represents the page, then these default values are used:
  
  ```
  regionalAreaWidth="256"
  isRegionalAreaCollapsed ="false"
  ```

- For programmatic control, drag and drop one of the following corresponding methods from the `FndUIShellController` data control.
  
  - `discloseRegionalArea`
  - `collapseRegionalArea`
  - `setRegionalAreaWidth`

Two APIs shown in Example 14-17 are exposed as data control methods under `FndUIShellController` data control.

---

**Note:**

- Declarative support allows the `inflexibleHeight` property to control the pixel height of the regional area panel. Programmatic support does not have this allowance.

- Programmatic support allows the `forceRefresh` property to make it possible to refresh a task without passing in any parameters. Declarative support does not have this allowance.

- Refreshing a regional area task without disclosing the task is not supported.

- Multiple regional area tasks are allowed to be disclosed at the same time. A switch to force showing only one task at a time is not provided.

- Support for persisting any of these settings explicitly altered by the user during a session, across sessions, is not a part of this feature.

---

**Example 14-16  Sample Menu File Entry**

```
<itemNode id="itemNode_SvcCenter"

label="#{adfBundle['oracle.apps.fnd.appcore.patterns.demo.patterns_demo_menuBundle'].SERVICE_CENTER]"

action="adfMenu_SvcCenter" focusViewId="/SvcCenter"

isDynamicTabNavigation="false" regionalAreaWidth="250"

isRegionalAreaCollapsed="false">
```
**Example 14-17  APIs Exposed as Data Control Methods Under FndUIShellController**

```java
/**
 * Discloses a regional area task.
 * @param taskFlowId Task flow to disclose
 * @param keyList Key list to locate the task flow instance.
 * This is a semicolon-delimited keys or key-value pair.
 * For example, "key1;key2=value2". If only the key is specified,
 * the value is picked up from the parametersList with the same
 * name as the key.
 * @param parametersList Parameters list for the task flow.
 * This is a semicolon-delimited String
 * of name-value pairs. For example,
 * "param1=value1;param2=value2".
 * @param label Label for the task flow
 * @param forceRefresh Default false. If false, task flow reinitialization
 * depends on whether some parameters are passed into parametersList, where
 * the presence of parameter values causes reinitialization and the absence of
 * parameter values does not. forceRefresh true always causes reinitialization
 * of the task flow regardless of the value for parametersList.
 * @return For internal Contextual Event processing
 */
public FndMethodParameters discloseRegionalTask(String taskFlowId,
                                                String keyList,
                                                String parametersList,
                                                String label,
                                                Boolean forceRefresh,
                                                FndMethodParameters methodParameters);
```

```java
/**
 * Collapses a regional area task.
 * @param taskFlowId Task flow to collapse
 * @param keyList Key list to locate the task flow instance.
 * This is a semicolon-delimited list of key-value pairs.
 * For example, "key1=value1;key2=value2".
 * @return For internal Contextual Event processing
 */
public FndMethodParameters collapseRegionalTask(String taskFlowId,
                                                 String keyList,
                                                 FndMethodParameters methodParameters);
```

### 14.4.3 How to Control the State of the Contextual Area Splitter

The UI Shell provides the means to control the disclosure state of the contextual area through either declarative or programmatic support.

**Declarative support** lets you specify the initial state when loading a work area JSPX page. It determines whether the contextual area (as a whole) is collapsed or disclosed.

**Programmatic support** lets you control the initial or subsequent state of the contextual area within a work area JSPX page.

- By default, the disclosure state is driven by what is specified declaratively. However, after the initial page load, you can override the declarative default and, for example, render the work area with the contextual area collapsed (overriding
the declarative setting of rendering that work area with the contextual area disclosed).

- Disclosing the collapsed contextual area splitter programmatically in response to a UI action by the user, such as a button click or menu selection.

**Samples of Expected Behavior**

- A work area JSPX page loads with the contextual area collapsed or disclosed when the page renders, based on the declarative setting. If the work area is loaded because of a Main Menu invocation, then declarative options always are used for the disclosure state.

- If a work area loads because of a page navigation from another work area, programmatically set options may override declarative settings.

### 14.4.3.1 Implement the Contextual Area Splitter

Follow these steps to implement the contextual area splitter:

1. Extend the `contextual-area-task-flow-template` task flow template into the page task flow, as shown in Example 14-18.

2. Specify values for the contextual area splitter position and the collapsed state in the menu for the item node that represents the page using the `contextualAreaWidth` and `contextualAreaCollapsed` properties. A sample entry in the menu file will resemble Example 14-19.

3. If these properties are not set in the menu for the top-level item node that represents the page, then these default values are used:

   ```
   contextualAreaWidth="256"
   contextualAreaCollapsed ="false"
   ```

4. For programmatic control, drag and drop one of the corresponding methods from the `FndUIShellController` data control:

   - `collapseContextualArea`
   - `contextualAreaWidthSelection`

5. To set these values when opening a new task, drag and drop the `openMainTask` method from `FndUIShellController` data control and pass in the `contextualAreaWidth` and `contextualAreaCollapsed` parameters through "methodsParameters > NamedData" as shown in Example 14-20.

   Set the method in the page managed bean to set the `contextualAreaWidth` and `contextualAreaCollapsed` values, as shown in Example 14-20.

6. To set these values using the Navigate API to navigate to a task flow, drag and drop the `navigate` method from `FndUIShellController` and pass in the `contextualAreaWidth` and `contextualAreaCollapsed` parameters through "methodsParameters > NamedData" as shown in Example 14-21.

   First you need to set the `FndMethodParameters`; this should be done by adding code similar to this to a managed bean that we will call `myBean` for this example.

   ```java
   private methodParameters fndMethodParams;
   ...
   public void myNavParameters()
   ```
FndMethodParameters methodParams = new FndMethodParameters();
HashMap testHashMap = new HashMap();
testHashMap.put("contextualAreaWidth", "200");
testHashMap.put("contextualAreaCollapsed", "true");
methodParams.setCustomObject(testHashMap);
fnMethodParams = methodParams;

Example 14-18    Extending the Task Flow Template

Example 14-19    Example of contextualAreaWidth and contextualAreaCollapsed
Properties

Example 14-20    Setting the contextualAreaWidth and contextualAreaCollapsed
Values for openMainTask

Example 14-21    Setting the contextualAreaWidth and contextualAreaCollapsed
Values for navigate
14.4.4 Sizing Regional Area Panels

Multiple regional area panels can be open at the same time, instead of showing only one panel at a time.

Because the desired size of each panel will be different, you can set the pixel height for each of the panels by specifying the `inflexibleHeight` property in the item node that represents a regional area panel, as shown in Example 14-22.

**Example 14-22 Using `inflexibleHeight` to Set Panel Height**

```xml
<itemNode id="__ServiceRequest__itemNode__SRSearch" focusViewId="/ServiceRequest" label="SR Search" taskType="defaultRegional" taskFlowId="/oracle/apps/fnd/applcore/patterns/demo/SRSearch.xml#SRSearch" inflexibleHeight="200"/>
```

14.5 Working with the Global Menu Model

Menu metadata used in Oracle Fusion Applications is divided into global menu data, consisting of the Home page tabs, the Navigator (also known as the Main menu), and the Preferences menu.

The Navigator and Home page tabs contain information from different applications, yet each application must be able to be developed independently. To bring this information together, a Global Menu Model is provided.

Navigation to a page is accomplished by constructing and executing a URL. Matching the application name from the distributed menu metadata to its deployment information will dynamically create the host/port portion of the URL. Other page parameters are held in the existing page-level menu metadata.

The Task Menu, create URL, and navigation API allow other declarative and programmatic access to page navigation. The UI Shell global area also will support a Home page link for page navigations.

**Global Menu Model Service**

This model:

- Contains at least the label, the application name, and the `viewID`
- Calls the Policy Store (optimized bulk authorization) to get the subset of these menu items to be rendered for that user
- [Restore post V1] Combines the results from all pillars and renders them
- Is distributed for multiple pillars
Example of Global Menus
The Global Menu Model presents a cascading appearance, shown in Example 14-27.

Global Menu Behavior

- Items to which the user does not have access will not be displayed.
- A category is hidden if there is no child to display.
- If a menu entry length is greater than 27 characters, ellipses (…) are displayed. The entire entry will be displayed in a tool tip when the pointer hovers over the entry.
- Parent and children categories will not be split into different columns.

14.5.1 How to Implement a Global Menu

Note:
Before you create menus, you first must create JSF pages using the UI Shell template.

These global menus span JavaEE applications.

- **Navigator**: This is the global menu that is displayed in the UI Shell global area.
- **Home Page Menu**: The Home page tabs are actually each a JSPX page assembled using menu metadata.
- **Preferences Menu**: The User Preferences page has a task list link to all other preferences pages within Oracle Fusion Middleware. This is assembled using menu metadata.

14.5.1.1 Menu Attributes Added by Oracle Fusion Middleware Extensions for Applications (Applications Core)

Table 14-3 and Table 14-4 list the menu attributes added by Applications Core to the menu XML in addition to what is provided by Oracle ADF.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>labelKey</td>
<td>xsd:string</td>
<td>N</td>
<td>Bundle key used for label; the key will be looked up in the resource bundle specified by the resourceBundle attribute of &lt;menu&gt;.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>webApp</td>
<td>xsd:string</td>
<td>Y</td>
<td>The webApp attribute is used to look up the host and port of the associated work area or Dashboard from the Oracle Fusion Applications Functional Core (ASK) deployment tables. These tables are populated at deployment time through Oracle Fusion Functional Setup Manager tasks.</td>
</tr>
</tbody>
</table>
Table 14-4  (Cont.) <itemNode> Attributes for Global Menus

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>focusViewId</td>
<td>xsd:string</td>
<td>N</td>
<td>This is the page ID. This can be found by looking in the adfc-config.xml file. The name under each page in the diagram view is the page ID.</td>
</tr>
<tr>
<td>securedResourceName</td>
<td>xsd:string</td>
<td>N</td>
<td>The resource name that is used for securing the item node.</td>
</tr>
<tr>
<td>applicationStripe</td>
<td>String</td>
<td>N/A</td>
<td>This attribute is used for pages. Check the security of the page against the policies that are located in LDAP. The applicationStripe name must be the same as the stripe name of the LDAP policy store, which is the same as the web.xml application.name attribute.</td>
</tr>
<tr>
<td>parametersList</td>
<td>String</td>
<td>N/A</td>
<td>This is a task-level itemNode attribute that is a parameters list to pass in to the task flow to open in the target workspace. This is a semicolon-delimited string of name-value pairs. For example, &quot;param1=value1;param2=value2&quot;</td>
</tr>
<tr>
<td>pageParametersList</td>
<td>String</td>
<td>N/A</td>
<td>This is a page-level itemNode attribute that is the parameters list for the page. This is a semicolon-delimited string of name-value pairs. For example, &quot;param1=value1;param2=value2&quot;. If the Expression Language expression evaluates to an object, the toString value of that object will be passed as the value of the parameter.</td>
</tr>
</tbody>
</table>
| destination        | String    | N/A      | The destination attribute is supported on the item nodes for the Navigator. The destination is necessary only for navigating to an external web site. When it is defined, it takes precedence over all attributes. Example of the menu data: <itemNode id="itemNode_otn" destination="http://www.oracle.com/technology/index.html"/>

14.5.1.2 Displaying the Navigator

The Navigator, shown in Figure 14-23, is rendered when the Navigator icon is clicked on the UI Shell.

Figure 14-23  Navigator Example

<table>
<thead>
<tr>
<th>Accounts Receivable</th>
<th>Expenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revenue</td>
<td>Auditing</td>
</tr>
<tr>
<td>Billing</td>
<td>Corporate Cards</td>
</tr>
<tr>
<td>Receivables Balances</td>
<td>Fixed Assets</td>
</tr>
<tr>
<td>Budgets</td>
<td>Additions</td>
</tr>
<tr>
<td>Budgets</td>
<td>Financial Transactions</td>
</tr>
<tr>
<td>Career</td>
<td>Retirements</td>
</tr>
<tr>
<td>Profile Content</td>
<td>Tracking</td>
</tr>
<tr>
<td>Cash Management</td>
<td>Analysis and Reporting</td>
</tr>
<tr>
<td>Bank Statements and</td>
<td>General Accounting</td>
</tr>
<tr>
<td>Reconciliation</td>
<td>Inquiry and Reporting</td>
</tr>
</tbody>
</table>

Items in bold are groups, and items in blue are links. So some groups also are links.
14.5.1.3 Implement a Global Menu

**Note:**
The Navigator is used as the example for how a developer implements a Global Menu, but the steps will be similar for the Preferences and Home menus.

The Navigator metadata may be pointing to target work area pages in various applications. To simplify the runtime behavior, one XML file contains all the menu entries. An Applications Core application will deploy these menus to Oracle Metadata Services (MDS). Each application will read these directly from MDS.

Each application must be configured so that the shared library can read the menus from MDS.

**To implement a Global Menu:**

1. Verify that the `web.xml` file of the application has the correct Java Authentication and Authorization Service (JAAS) filter to enable checking menu security against Oracle Platform Security Services (OPSS), as shown in Example 14-23.

   The `application.name` parameter, as shown in the example, in the `web.xml` file is the application family value. The choices are crm, fscm, and hcm. This value is used to create the stripe in LDAP.

2. Update the `weblogic-application.xml` file. As shown in Example 14-24, set the `application-param` that has the `param-name` `jps.policystore.migration` to `OFF`.

3. In the `weblogic-application.xml` file, ensure that the `application-param` that has the `param-name` `jps.policystore.applicationid` is set to the correct stripe, as shown in Example 14-25. This is the same as the `application.name` property of the `web.xml` file.

4. Add the following entry in the `web.xml` file:

   ```xml
   <listener>
   <listener-class>oracle.apps.fnd.applcore.menu.service.MenuFragmentServiceContextListener</listener-class>
   </listener>
   
   Example 14-23   Sample JAAS Filter
   
   ```xml
   <filter>   
   <filter-name>JpsFilter</filter-name>   
   <init-param>   
   <param-name>enable.anonymous</param-name>   
   <param-value>true</param-value>   
   </init-param>   
   <init-param>   
   <param-name>remove.anonymous.role</param-name>   
   <param-value>false</param-value>   
   </init-param>   
   <init-param>   
   <param-name>application.name</param-name>   
   ```
Example 14-24 Setting jps.policystore.migration to OFF

Example 14-25 Setting jps.policystore.applicationid to the Correct Stripe

14.5.2 How to Set Up Global Menu Security

Global menu security depends on applications using a standalone LDAP server.

Note:

Global security only works with standalone WebLogic Server.

ADF Menu Security is enabled by default. If you need to disable menu security, such as for testing, start WebLogic Server after setting the JAVA_OPTIONS environment variable in the setDomainEnv.sh file:

JAVA_OPTIONS = -DAPPLCORE_TEST_SECURED_MENU=N

14.5.2.1 Enforcing User Privileges and Restrictions

Before you enforce user actions, define roles, principals, and actions in the database. Functional security will always prevent a user from accessing a page or task flow for which the user does not have privilege. To improve the user experience, global menus can be hidden if the user does not have access to that page. There are two different security features for this:

- The global menus have a securedResourceName attribute, which should be the value of the page resource against which security can be checked. For pages, this is the page definition file.

- The menus also have a rendered attribute. This can be used to evaluate an Expression Language security expression. If rendered="false" (false being the outcome of the expression), the menu item will be hidden even if the user has access to the page. There are certain times you would want to do this. For instance, consider a person working in HR as a consultant, not an employee. You might want a menu entry for editing employee data under an HR category, without showing an entry under the Employee Self-Service category that also leads to the same page. See Example 14-26.
The Expression Language expression should never check the page definition. However, you can use the Expression Language expression to check security of a person’s role because that is in LDAP.

- The applicationStripe attribute determines which LDAP stripe is checked for the securedResourceName attribute.

Example 14-26  Expression Language Expression to Evaluate a User’s Access Rights

rendered="#{securityContext.userInRole['EMPLOYEE_ROLE']}"

14.5.3 How to Create the Navigator

Menu files will be referenced through MDS. This means they can be located in a table or in a file system directory. Determine where you want this directory to be located. This is where your root_menu.xml file and other menu files will be located. For Global menu attributes, see Menu Attributes Added by ()

1. Create the Navigator.

   Example 14-27 shows a sample Navigator.

2. Create the application's Navigator files.

   The next files in the menu hierarchy can contain groupNodes that appear as non-clickable categories, itemNodes that are clickable to start a page, or references to more menu files. If itemNodes were included that were not deployed, they will not appear because what was deployed is checked against the deployment tables. Applications Core requires that if a groupNode has no child nodes, which could happen through security enforcement, the groupNode itself will not be rendered.

Example 14-27  Example of a Navigator

<menu xmlns="http://myfaces.apache.org/trinidad/menu">
  <groupNode id="groupNode_my_information"
    idref="_groupNode_my_information_
    label="#{menuBundle['oracle.apps.menu/ResourcesAttrBundle'].MY_INFORMATION}"
    securedResourceName="oracle.apps.hcm.people.portrait.ui.page.MyPortraitPageDef"
    applicationStripe="hcm"
  
    <itemNode id="itemNode_my_information_my_portrait"
      label="#{bundleVar.MY_PORTRAIT}" focusViewId="/MyPortrait" webApp="HcmCore"
      securedResourceName="oracle.apps.hcm.people.portrait.ui.page.MyPortraitPageDef"
      applicationStripe="hcm"
    
    <groupNode id="groupNode_my_information_compensation"
      idref="_groupNode_my_information_compensation_
      label="#{menuBundle['oracle.apps.menu/ResourcesAttrBundle'].COMPENSATION}"
      securedResourceName="oracle.apps.hcm.goals.core.publicUi.page.ManageGoalsWorkAreaPageDef"
      applicationStripe="hcm"
    
    <itemNode id="itemNode_my_information_goals"
      label="#{menuBundle['oracle.apps.menu/ResourcesAttrBundle'].GOALS}"
      focusViewId="/ManageGoalsWorkArea" webApp="HcmTalent"
      securedResourceName="oracle.apps.hcm.goals.core.publicUi.page.ManageGoalsWorkAreaPageDef"
      applicationStripe="hcm"
    
    <groupNode id="groupNode_my_information_career"
      idref="_groupNode_my_information_career_
      label="#{menuBundle['oracle.apps.menu/ResourcesAttrBundle'].CAREER}"
      securedResourceName="oracle.apps.hcm.goals.core.publicUi.page.ManageGoalsWorkAreaPageDef"
      applicationStripe="hcm"
    
    <itemNode id="itemNode_my_information_performance_management"
      label="#{menuBundle['oracle.apps.menu/ResourcesAttrBundle'].PERFORMANCE}"
      applicationStripe="hcm"
      focusViewId="/PerformanceWorkArea" webApp="HcmTalent"
    
</menu>
14.5.3.1 Render the Navigator as Dropdown Buttons

There are situations, particularly with simpler applications, when the default enterprise-level menu structure is not suitable. In these cases, you may want to display the Navigator as a series of dropdown buttons.

To switch the UI Shell rendering so the Navigator renders as dropdown buttons in a horizontal row, set the `isSelfService` attribute to "true" on the `.jspx` page that extends the UI Shell template. That is, inside the `<af:pageTemplate>` tag, add the following:

```html
<f:attribute name="isSelfService" value="true"/>
```

14.6 Using the Settings and Actions Menu

The Settings and Actions menu, shown in Figure 14-24, is accessed by clicking the name of the logged-in user on the right edge of the global area. It consolidates several menus that previously were stand-alone.
• **Sign In / Sign Out**

**Note:** When signing in, users *always* are directed to the application's home page.

This list provides two possible scenarios during runtime:

1. The application is not secured. In this case, there is no concept of the user being logged in (authenticated) or logged out (not authenticated).
   - The commandLink displays the text Sign In and is disabled.

2. The application is secured with ADF Security.
   - If the user is logged in:
     - The commandLink displays the text Sign Out and is enabled.
     - If Oracle WebLogic Server is configured to authenticate with Oracle Internet Directory LDAP, the user name is the display name of the authenticated user principal. The display name is indexed by a general end user preference. See [How to Implement End User Preferences](#).

If the user is logged out, the only way for an unauthenticated user to view a page is if a page either has no databinding (no pagedef) or has databinding but is granted to the anonymous role:

   - The commandLink displays the text Sign In and is enabled.

When the **Sign Out** link is clicked, the page is redirected to the application’s default home page if it is accessible. There are cases in which a default home page is not available to the current user. In such cases, the user application should implement the `setLogoutUrl()` API that is included in the `oracle.apps.fnd.appcore.common.ApplSession` class of the Oracle.
Fusion Middleware extensions for Applications Core API. This API is implemented as:

```java
public void setLogoutURL(String pLogoutURL);
```

The UI Shell’s Sign Out logic will get the logout URL from ApplSession using the `getLogoutURL()` API and redirect to it. This API is implemented as:

```java
public String getLogoutURL();
```

If the logout URL was not set in ApplSession, the application will be redirected to the default home page.

On clicking the Sign Out link, the user session is cleared from the cookie and terminated. If the home page is secured, the login prompt will first appear. If the home page is not secured, the page will appear and the Sign In link will be enabled.

- **Switch To / Impersonation**

  The Switch To menu is available only if the user has the security privilege to impersonate others and has been given access to impersonate at least one other user. When clicked, the menu displays the name of the logged-in user and any user names he or she can impersonate.

  ![Figure 14-25 Switch To Menu](image)

  See How to Set Up a Proxy User (Session Impersonation).

**14.6.1 How to Use the Personalization Options**

The options in the Personalization section of the Settings and Actions menu let you set your preferences, edit the current page, and reset the content and layout.

The Set Preferences option only appears if you have the ApplSession filter and mapping set up. See Configuring Your Project to Use Application User Sessions.

**Set Preferences**

You create the actual Preferences dialog, such as shown in Figure 14-28. See How to Implement End User Preferences for the details of how to implement the Preferences menu.

**Edit Current Page**

This option is displayed only if the displayed page has been marked as able to be user-edited (if the isPersonalizableInComposer attribute in af:pageTemplate is set to true). Selecting this option will start the editing feature and the page will resemble Figure 14-26. Click Close to return to the page. Click Customization Manager to change the displayed page in Oracle Composer. For more information about the Customization Manager, see the Oracle Fusion Middleware Developing WebCenter Portal Assets and Custom Components with Oracle JDeveloper guide. Note that changes are for just this user and therefore are called personalization. See Creating Customizable Applications.
Reset to Default Content and Layout
Select this option to discard any personalization changes and return to the default settings. Note that resetting layout and content is for that page. In particular, if any task flows are personalized on that page using Oracle Composer, they are not reset by this menu item.

When you select this option, this confirmation dialog is displayed:

**Figure 14-27  Confirm Reset Content and Layout**

---

### 14.6.2 How to Implement End User Preferences

The Set Preferences option provides easy access to setting preferences for the current application, general user preferences, or for any other application preference in Oracle Fusion Middleware. For more information about this Global menu, see *Working with the Global Menu Model*.

Preferences are pages that can set system wide settings that applications can access. There are general preferences that affect all applications, and there can be applicationspecific preferences. General preferences include language, date format, and currency. General preferences are stored in LDAP so they can be accessed from any application. Application preferences are usually stored in the applications on database tables, but they can be stored in LDAP also.

#### 14.6.2.1 Using the Set Preferences Option

Click **Set Preferences** to open a work area that shows the preferences related to the currently displayed page.

Links in the left hand side will allow navigation to any Preferences work area page within the entire Oracle Fusion product. This page will be rendered using Applications Core menu federation abilities. As the developer, you will own the menu files.
If an application is not installed, or the user does not have access, the entry in the task list menu should not appear. If a user does not have access to a particular setting within a page, you must use the rendered property with a security expression behind it.

For each application, there should be a preferences page. The preferences page uses the same path of the current application, but it uses a page name of preferences.jspx.

If no associated preferences page exists, a default General Preferences page will be shown. This page shows global Applications Core most-used preferences.

If there are several preference pages associated with an application, such as a Common Setting page and more specific pages, only one preferences page as a target from the global preferences link can be defined per application. (There will be a default name for the target focusViewID of the preferences page for an application.) In the preferences work area, other links are available from the task list to more specific pages or task flows. (Links in the task list can contain other focusViewIDs that belong to the same application as the default preference page.)

When the first application is deployed, it will become the location of the General Preferences page.

Several pages of an application can all point to the same preferences page.

More than one application cannot point to the same preferences page. This implies that each application can have its own preferences page, and if two applications want to share a common preferences page, they can, but it can be navigated to only from the task list. Therefore, from the Preferences link, the user always sees the more specific preferences page of that application.

**14.6.2.2 Using the Preferences Work Area Page**

A preferences page will be like any other work area page. Preference values are not supported in integrated Oracle Weblogic Server LDAP, only an external LDAP is supported. The task list will be loaded as a defaultRegional task flow and the main area will be a defaultMain task flow.

**Work Area Title**

Each preferences page should display a title similar to {Category_name:Page_name}. This can be done through an Expression Language expression and will not be created automatically from the framework.

The name that appears in the task list can be different from the page title. This is allowed because the task list name is generated from the task list preference distributed menu metadata, while the page title will be from the local page-level menu metadata.

**Tasklist and Navigation Pane**

Each page needs its Application Menu metadata to specify that it wants the Preferences task list menu in the defaultRegional area as well as in the defaultMain task flow.

This menu can be a two-level menu having categories with links under each category.

**Task List Federation**

The task list will be a task flow that will contain links to all preference pages throughout Oracle Fusion Middleware.
Each application will provide the preferences menu files that contain task list links to preferences pages delivered by that application. The preferences task list should follow the Navigator architecture recommendations where it uses sharedNode references to bring in menus from each application so they can be patched independently. Applications Core will automatically federate the menu metadata so the task list that is rendered will contain all the entries from all applications (filtered by security).

Individual menu files will have versions like other distributed menu files, so any application can apply a patch and the new menu will take precedence over an older version when federated.

**Task List Can Link Only to Full Pages (Not to Specific Task Flows)**

The task list will not start task flows dynamically, but will load a Preferences workarea page. This is because the task list menu must be federated and only page-level entries are allowed in a federated menu.

**No-Tabs Mode**

The Preferences page should use a no-tabs mode. This is a standard, not controlled through any code. You can use tabs if all task flows are defaultMain. See Supporting No-Tab Work Areas for more information.

**Task List Security**

The task list will be filtered by functional page-level security for that user. If all entries in a category are restricted, then the category should not appear either.

**Preferences Settings**

Settings will be a view activity in a task flow. It will follow other user experience standards so it should be built using an Applications Panel. This means the action buttons will appear at the top.

Different preferences pages can change the same backend setting, depending upon applications design. If this is needed, it should be stored in a common area, such as LDAP, or be in the General Preferences page.

**14.6.2.3 Deploying Preferences Pages and Designing General Preferences Content**

Application preferences pages are deployed with the corresponding product pages. The design should be similar to that shown in Figure 14-28.

**Figure 14-28  General Preferences Example**

![General Preferences Example](image)

**14.6.2.4 Configure and Implement End-User Preferences**

After the WebLogic Server console is configured, follow these steps to create an Oracle Fusion web application that uses UI Shell pages:
1. Create a UI Shell page that is used solely for the user preferences, such as PreferencesUI.jspx.

2. Set the isDynamicTabNavigation property to false for the PreferencesUI page entry in the menu.

3. Add the following task flow as a default regional area under the preferences page entry: 
   "WEB-INF/oracle/apps/fnd/applcore/pref/ui/mainflow/GeneralPreferencesFlow.xml#GeneralPreferencesFlow"

   The final menu entries for the page will appear similar to those shown in Example 14-28.

   The results of this example should display the basic Preferences menu entries in the default regional area that can be opened to display subflows for each preferences subtask (for instance, Accessibility and Appearance).

Example 14-28 Sample General Preferences Menu Entries

```xml
<itemNode id="itemNode_untitled2" label="Preferences"
    action="adfMenu_PreferencesUI" focusViewId="/PreferencesUI"
    isDynamicTabNavigation="false" webApp="Demo">
    <itemNode id="def1" focusViewId="/PreferencesUI" label="General Preferences"
        taskType="defaultRegional"
        taskFlowId="/WEB-INF/oracle/apps/fnd/applcore/pref/ui/mainflow/GeneralPreferencesFlow.xml#GeneralPreferencesFlow"/>
</itemNode>
```

14.6.2.4.1 How to Use the Preferences Menu Model

The General Preferences task flow that is exposed also renders the Preferences Menu Model links by using a call to the Menu Service API.

The Preferences menu will be part of a central Utility application (menu web service) that will be deployed in the server. As a developer, you maintain the Preferences menu.

On any UI Shell page, the global area contains a Personalization menu that contains a Set Preferences link. This link will redirect the user to an application-specific Preferences page, depending upon the entry in the menu data.

Example 14-29 shows sample Preferences menu data.

In the example menu XML file, each parent item node represents a Preferences UIShell page. Its child nodes refer to the application-specific task flow in which the Preferences page exists.

For example, the first itemNode entry refers to the preferencesA page that is part of the FND webApp. The Service Flow child node is a task flow that belongs to the FND webApp.

For each parent itemNode, there is an attribute called prefForApps that contains a list of webApp names. This means that the itemNode is a common preferences page for those listed webApps.

For example, the Preferences page is common for two webApps -- gl and hr. This essentially means that all the Dashboards and work area UI Shell pages in the gl and hr webApps will be redirected to this preferencesA page, which is in webApp FND, when the Set Preferences link is clicked.
All the task flows under each preferences page `itemNode` will be displayed in the General Preferences task flow as navigation links. Therefore, all preferences pages will have access to these task flows.

**Example 14-29  Example Preferences Menu Data**

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<menu xmlns="http://myfaces.apache.org/trinidad/menu" version="1">
  <itemNode id="preferences_node_a" label="Preferences Page A" action="preferences_node_a" focusViewId="/preferencesA" webApp="fnd" prefForApps="gl, hr">
    <itemNode id="Flow 1" label="Service Flow" focusViewId="/preferencesA" taskFlowId="/WEB-INF/oracle/apps/fnd/applcore/finance/ServiceFlow.xml#ServiceFlow" parametersList="id=userName" />
  </itemNode>
  <itemNode id="preferences_node_b" label="Preferences Page B" action="preferences_node_b" focusViewId="/preferencesB" webApp="fnd" prefForApps="fn">
    <itemNode id="Flow 2" label="Request Flow" focusViewId="/preferencesB" taskFlowId="/WEB-INF/oracle/apps/fnd/applcore/finance/RequestFlow.xml#RequestFlow" parametersList="id=userName" />
  </itemNode>
  <itemNode id="preferences_node_d" label="Preferences Page D" action="preferences_node_d" focusViewId="/PreferencesUI" webApp="Demo" prefForApps="Demo">
    <itemNode id="Flow 2" label="Request Flow" focusViewId="/preferencesB" taskFlowId="/WEB-INF/oracle/apps/fnd/applcore/finance/AcceptanceFlow.xml#AcceptanceFlow" />
  </itemNode>
</menu>
```

14.6.2.4.2 How to Configure User Session and ADF Security

To test the general preferences task flows, configure user session and ADF Security for the test application. See Implementing Application User Sessions.

When configuring ADF Security, there is no need to define users, because you already are using an Oracle Internet Directory store that will authenticate the users existing in the policy store.

14.6.2.4.3 How to Retrieve Preference Values and Check Accessibility Mode by Using an Expression Language Expression

See Working with Localization Formatting.

A use case exists where the UI must use an Expression Language expression to check whether the accessibility mode is set to `screenReader` to render screen reader-friendly components in screenReader mode. The recommended method to do this uses `#{requestContext.accessibilityMode}` and is documented in the How to Configure Accessibility Support in trinidad-config.xml section of the Fusion Middleware Developing Web User Interfaces with Oracle ADF Faces guide.

14.6.2.4.4 How to Implement the Password Management Page

The Password link on the General Preferences page will point to the Password Management page from the Oracle Identity Management administration application, as shown here:
For the Password link to redirect to the pwdmgmt.jspx page, the deployment information of the current application and the Oracle Identity Management administration application must be populated correctly in the Oracle Fusion Applications Functional Core (ASK) tables. You may need to talk to the person or team responsible for maintaining this information.

### 14.6.2.5 Using the Most Common Preferences

The Preferences most end users want to set are Language, Accessibility, and Regional. This section discusses how to configure these Preferences and what the default rules are, such as what happens if you do not set the Language Preference.

See [Working with Localization Formatting](#) for more information about localization of Preferences.

#### 14.6.2.5.1 How to Configure the Language Preference

When a session is created, it will use the language provided in the ORA_COMMON_PREFS cookie if it is set to a valid language. Inside the cookie, the fusionapps_language attribute is used to determine the language. There also is a local attribute that is for non-Oracle Fusion applications so that Oracle Fusion applications have their own attribute in the cookie and allowing other applications to keep their own language preference. Oracle Identity Management knows which to use based on a system value oracle.fusion.appsmode that is set at provisioning time.

If it is not a valid language or if the cookie does not exist, the browser language is used.

If for some reason the browser language is not valid, the seeded FND_LANGUAGE profile option is checked to determine the default language, which defaults to American English. If accessing a protected page, the login screen appears. This login screen may allow the user to select a language for the session. If selected, this will override the language. If the user does not select anything on login and if the user has a language preference previously saved, this will override the language if it is an installed language. If saved, it would be held in the LDAP settings for the Oracle Fusion applications Language setting in the orclFALanguage attribute of the orclIDXPerson object. This is controlled by Oracle Identity Management.
Users can set the Language preference by selecting the **Set Preferences** link from the Personalization menu in the global area. The Language option will look similar to Figure 14-30.

**Figure 14-30  Setting the User Preference Language Option**

14.6.2.5.2 How to Configure the Accessibility Preference

Oracle Fusion applications follow the Apache Trinidad standards for Accessibility.

Users can set the Accessibility preference by selecting the **Set Preferences** link from the Personalization menu in the global area. The Accessibility option will look similar to Figure 14-31.

**Figure 14-31  Setting the Accessibility User Preference Options**

14.6.2.5.3 How to Configure the Regional Preferences

The default Regional settings are based on the selected Language setting unless the Regional setting is saved explicitly. That is, no matter how the current language is derived, whether it is from login, the browser or LDAP, if the Regional settings are not yet set, they will default from the language. But if the user sets the Regional settings and then sets the Language setting, the Regional settings will not be changed.

Users can set the Regional preferences by selecting the **Set Preferences** link from the Personalization menu in the global area. The Regional options will look similar to Figure 14-32.

**Figure 14-32  Setting the Regional User Preferences Options**
14.6.3 How to Use the Administration Options

The Administration options in the Settings and Actions menu are displayed only if the logged-in user has the appropriate privileges. See Securing the Administration Menu.

Customize [name_of_page] Pages...
Select this option to customize the current page for multiple users using the customization layer picker dialog.

For information about customization, see Creating Customizable Applications.

Customize Global Page Template
Select this option to open the page in composer mode. All the global items can be selected so you can customize them. Customization is done at the site layer so it will apply to all pages, not just the one you are on. (When you use Customize [name of page] Pages, all the content except the global content is available for customization.)

Manage Customizations...
Select this option to start the Customization Manager.

For information about customization, see Creating Customizable Applications. For more information about the Customization Manager, see the Extending Runtime Editing Capabilities Using Composer section and the "Introduction to Composer" section of Developing WebCenter Portal Assets and Custom Components with Oracle JDeveloper.

For information about defining and configuring namespaces when promoting a page fragment to a label, see "Updating Your Application’s adf-config.xml File" in the "Performing Composer-Specific MDS Configurations" section of Developing WebCenter Portal Assets and Custom Components with Oracle JDeveloper.

Manage Sandboxes...
Select this option to manage sandboxes on your system.

The Sandbox is built on top of the standard Sandbox feature from Oracle Metadata Service.

Setup and Maintenance...
Select this option to start the Oracle Fusion Functional Setup Manager application. See the Oracle Fusion Functional Setup Manager User’s Guide.

Highlight Flexfields
No separate implementation is required. This option is available to any user who has the permission to view the Administration menu.

Select this option to highlight the Descriptive Flexfields (DFF), Key Flexfields (KFF) and Extensible Flexfields (EFF) on the runtime page.

When this mode is on, the page displays an information icon by each DFF, KFF and EFF. This will be displayed whether any Flexfield segments have been configured.

Hovering over the icon shows the Flexfield details. Administrators can note the DFF, KFF or EFF code so they can search for it in the Manage Descriptive Flexfields, Manage Key Flexfields, and Manage Extensible Flexfields setup tasks in Oracle Fusion Functional Setup Manager.

The option changes to Unhighlight Flexfields so that administrators can return to the normal view of the page.
For more information about Flexfields, see Getting Started with Flexfields.

14.6.3.1 Securing the Administration Menu

If you need the Administration link, include the privilege and the permission in the JAZN file as defined in Example 14-30. All administrator roles must inherit the Applications Core "Administration Link View Duty" duty role. This duty role gives access to the "View Administration Link" privilege.

Example 14-30  Required Privilege and Permission in JAZN File

```xml
<app-role>
  <name>FND_ADMINISTRATION_LINK_VIEW_DUTY</name>
  <display-name>Administration Link View Duty</display-name>
  <description>Provides access to the Administration Link on the UI Shell</description>
  <guid>EA1D0BF0BC096F11B18BDEB05F4BD0B48</guid>
  <class>oracle.security.jps.service.policystore.ApplicationRole</class>
  <members>
    <member>
      <class>oracle.security.jps.internal.core.principals.JpsXmlEnterpriseRoleImpl</class>
      <name>FND_APPLICATION_DEVELOPER_JOB</name>
    </member>
    <member>
      <class>oracle.security.jps.internal.core.principals.JpsXmlEnterpriseRoleImpl</class>
      <name>FND_APPLICATION_ADMINISTRATOR_JOB</name>
    </member>
  </members>
</app-role>

Privilege

```xml
<app-role>
  <name>FND_VIEW_ADMIN_LINK_PRIV</name>
  <display-name>View Administration Link</display-name>
  <description>Privilege to view administration link in UI shell. This privilege is available from Roles(s): Supply Chain Application Administrator,Cost Accountant,Application Implementation Consultant,Application Developer,Application Administrator</description>
  <guid>B14A4E74ECC63A3C6E4AF95816474D</guid>
  <class>oracle.security.jps.service.policystore.ApplicationRole</class>
  <members>
    <member>
      <class>oracle.security.jps.service.policystore.ApplicationRole</class>
      <name>FND_ADMINISTRATION_LINK_VIEW_DUTY</name>
      <guid>EA1D0BF0BC096F11B18BDEB05F4BD0B48</guid>
    </member>
  </members>
</app-role>

Permission and Grant to be included in JAZN file

```xml
<grant>
  <grantee>
    <principal>
      <class>oracle.security.jps.service.policystore.ApplicationRole</class>
      <name>FND_VIEW_ADMIN_LINK_PRIV</name>
      <guid>B14A4E74ECC63A3C6E4AF95816474D</guid>
    </principal>
  </grantee>
</grant>
```
14.6.4 How to Use the Troubleshooting Options

The Troubleshooting options provide user access to the standard help system and to troubleshooting and diagnostic tools.

Run Diagnostics Tests

Selecting this Settings and Actions menu Troubleshooting option opens the Diagnostics Dashboard user interface in a new window. For more information, see "Understanding Oracle Fusion Applications Incidents and Log Files" in the Administrator’s Guide.

Troubleshooting Options

Select this option to display the Options dialog, as shown in Figure 14-33.

**Figure 14-33  Troubleshooting Options Dialog**

- **Enable all**: Select this option to enable all other options on the dialog. When Enable all is selected, removing the selection from any of the other check boxes will deselect Enable all.

Applications logging, severity level and modules are stored as user-level profile options in profile tables. The corresponding profile option names are AFLOG_ENABLED, AFLOG_LEVEL, and AFLOG_MODULE. See “Configuring Settings for Log Files and Incidents” in Administrator’s Guide for information.

Because applications logging, severity level, and modules are profiles, when users click Save and Close, user-specific profile values will be inserted into the profiles. If users decide to revert the setting to the default site profile, they must use the Functional Setup Manager to remove their own profile.

After making changes to any one of the options for applications logging, severity level, or modules, the user must log out of the Oracle Fusion application, close the browser session, and log back in for the new options to take effect. These logging profiles are cached in the user session and initialized when a user logs into an Oracle Fusion application.
• **Database trace:** This option enables the SQL trace feature for all database connections used by the current user session.

For SQL trace, the trace file will have the FND session ID appended to the end. For example, `mysid_ora_4473_881497BF7770BE6604E040A0D807881.trc`.

The trace file can be found on the database host in the directory specified by the `user_dump_dest` init.ora parameter.

  – **Capture bind variables:**
    
    Select this option to enable the SQL trace option to also capture bind variables.

  – **Capture wait events:**
    
    Select this option to enable the SQL trace option to also capture wait events.

• **PL/SQL profiler:** This option enables the PL/SQL hierarchical profiler for all the connections used by the current user session.

For the PL/SQL profiler, the output will be in the directory defined by `APPLLOG_DIR`. The exact path for `APPLLOG_DIR` can be found on the database host by using the SQL statement:

```
select directory_name, directory_path from dba_directories
where directory_name like 'APPLLOG%'
```

The file names would be `PLS_<some number>_<FND session id>_<timestamp>.txt`, such as `PLS_49774696_88740EC94E3A0D2CE040A0D8036D8_100607104716.txt`.

To process the collected PL/SQL profiles and view results, run the `plshprof` command under the `$ORACLE_HOME/bin` directory.

• **Applications logging:** Applications logging is selected by default. Disabling logging will warn users that no logging will take place.

  – **Severity Level:**
    
    Use this option to set what types of information to log, and how much of it to log.

    For more information, see "Understanding Oracle Fusion Applications Incidents and Log Files" in the *Administrator’s Guide*.

  – **Modules:**
    
    This is the module filter for logging. This is a comma-separated list of modules to be logged. The percent sign (%) is used as a wildcard. For example, `% or %financial%`. The percent sign (%) is the default value and if no other value is specified, then it means everything will be logged.

    When a customer logs a service request with Oracle, the support person will help the customer enter the values necessary to filter the diagnostic logs for the needed information.

**Tracing Options**

This feature is designed for customers to collect diagnostic data to send back to Oracle for analysis.
When there is a performance issue that is reproducible, the tracing feature streamlines
the data collection process by allowing the end user who is experiencing the problem
to record a trace that will contain the necessary diagnostic information.

Before the user can access this function, the security administrator needs to assign the
Business Flow Troubleshooting role to the user. On each managed server, a maximum
of three users can be tracing at a time. By requiring the troubleshooting role to be
granted, an administrator can grant access to users who actually need to trace, and
revoke access when the issue is resolved.

To grant the role, the administrator will follow these basic steps:

- Log in to Oracle Identity Management.
- Click Administration on top right.
- Search for and select the target user.
- Click the Roles tab.
- Click Assign.
- Search for the display name Business Flow Troubleshooting.
- Select the role in the search results and click Add.

To run the trace, the user will follow these steps:

- Start the trace.

  The trace should be started only after navigating to the target managed server
  where the issue is happening. Note that the tracing feature only works within a
  single managed server; it will not span a flow involving more than one
  application (unless both applications happen to be on the same managed server).

  - Navigate to the target application.
  - Right before starting the problematic flow, select Troubleshooting > Tracing
    Options. The dialog shown in Figure 14-34 is displayed.

    Figure 14-34 Initial Tracing Options Dialog

    - Depending on the type of issue, select Performance or Functional trace type.
      The selection affects what diagnostic information is captured.

      In general, the user would select Performance when the problem is an
      application running too slowly, and Functional when the application does not
      behave as expected or when the information displayed is incorrect.
Trace Options defaults are based on the Trace type selection. Click Advanced Options to see what trace settings are enabled. For fine-grained control, select different trace options, as shown in Figure 14-35.

**Note:**
The user should accept the default settings or contact Oracle support for additional instructions.

---

**Figure 14-35  Advanced Tracing Options Dialog**

- Click **Start Trace**. A confirmation dialog, similar to Figure 14-36, with the trace ID will be displayed. Make a note of the trace ID.

---

**Figure 14-36  Trace Start Confirmation**

Subsequent clicks will be traced. The trace will run for a maximum of 30 minutes. If there are more than three simultaneous tracing sessions, an error message will be displayed.

If the Applications Trace option is selected (enabled by default for the Performance trace type), the server also needs to have at least 200MB of free memory in the JVM heap.

- Run the flow to reproduce the issue.
- Stop the trace.
– After executing the steps that have issues, select **Troubleshooting > Tracing Options** again. When tracing is active, only **Stop Trace** and **Cancel** will be active, as shown in Figure 14-37.

*Figure 14-37  Stop Trace*

![Stop Trace](image)

– Click **Stop Trace**. Figure 14-38 shows the two options to control whether to create an incident.

*Figure 14-38  Select Stop Trace Option*

![Select Stop Trace Option](image)

If the issue is reproduced, select **Stop trace and create incident**.

Click **OK**. A confirmation will be displayed with the incident number. Note the incident number in addition to the trace ID captured earlier.

The collected diagnostic information will be captured in an incident with the given incident number. See the “Troubleshooting Oracle Fusion Applications Using Incidents, Logs, QuickTrace, and Diagnostic Tests” chapter in the *Administrator’s Guide*.

If **Stop trace and do not create incident** is selected, then the confirmation will not have an incident number.

**User Productivity Kit**

The User Productivity Kit (UPK) is an optional, separately-purchased application that provides a content development, deployment, and maintenance platform for increasing project, program, and user productivity. The menu option will be available if Oracle UPK has been purchased, installed and configured.

This context-param entry must be added to the `web.xml` file:

```xml
<context-param>
    <description>This parameter notifies ADF Faces that the ExecutionContextProvider service provider is enabled. When enabled, this will start monitoring and aggregating user activity information for the client initiated requests. By default this param is not set or is false.</description>
    <param-name>oracle.adf.view.faces.context.ENABLE_ADF_EXECUTION_CONTEXT_PROVIDER</param-name>
    <param-value>true</param-value>
</context-param>
```

For recording, it may be necessary to turn on automation if all client components need to be sent down. The following parameter should be present in the application
web.xml:

<context-param>
<description>
This parameter notifies ADF Faces that test automation is being used. When enabled, this will cause the ids of components with testId attributes to be set to the value of the testId and the client component attribute of the component to be forced to true.

TestId attribute is now deprecated; use the 'id' attribute instead.
</description>
<param-name>oracle.adf.view.rich.automation.ENABLED</param-name>
<param-value>true</param-value>
</context-param>

Also, "upk" must be provisioned by the System Administrator. That is, an entry with DEPLOYED_MODULE_NAME = "upk" must be added in the ASK deployment tables. These tables are populated at deployment time through Oracle Fusion Functional Setup Manager tasks. Without the entry, the menu item "User Productivity Kit ..." will not be shown in the Help menu.

**Applications Help**
Select this option to open the help system in a separate window.

**Privacy Statement**
Select this option to display the privacy statement, which will appear in a new browser window. This option is always inactive until it is implemented. To set up the privacy statement, enter a fully-qualified URL in the PRIVACY_PAGE profile option.

Open the Setup and Maintenance work area, and use the Manage Applications Core Administrator Profile Values task to find the Privacy Statement URL profile option. In the Profile Value column, enter the full URL of the web page that contains the privacy content you want the menu item to link to.

Also see About Setting and Accessing Profile Values.

**About Applications**
Select this option to display the Oracle copyright statement and information about the application.

**About This Page**
Select this option to display version information for Oracle Fusion Applications, including details of the Oracle Application Development Framework, Oracle Metadata Services, the database, and the Applications Core. This makes it easier to know to which product the current page belongs and makes it easy to log a Service Request or find a patch for the correct product.

**Note:** This information is visible only to users with specific privileges. The privileges are controlled by the profile option FND_VIEW_VERSION_INFORMATION_PRIV, which an Administrator can enable or disable.

**14.6.5 How to Set Up a Proxy User (Session Impersonation)**

Session impersonation is a feature that enables a user (the impersonatee) to allow other users (impersonators) to log in and act as himself without sharing his password to the other user. The impersonator can login as himself and switch to the impersonatee. Once switched, he can access all the roles of the impersonatee. This
switch will be valid only for the current session; it must be done again in the next login. The impersonator can switch back to himself at any time.

**Note:**

Only users who have the `FND_IMPERSONATE_USER_PRIV` privilege will be available for the impersonatee to grant impersonation. The administrator must associate this privilege to the users who can serve as proxy users and can revoke it at any time. The grant is stored in LDAP as part of the impersonatee user attribute, `orclImpersonationGrantee`.

To set up the impersonation, follow these steps.

1. Log in as user Impersonatee (the user who will be impersonated by an authorized user).
2. Click your user name to open the Settings and Actions menu.
3. Click Set Preferences…
4. In the Tasks list, select Proxies.

*Figure 14-39  Selecting the Proxies Task*

The Proxies pane is displayed.

*Figure 14-40  Proxies Pane*

5. In the Proxies pane, click the Create icon to open the Select and Add: People dialog.

*Figure 14-41  Initial Select and Add: People Dialog*
6. Enter known data in any of the fields and click **Search**. This example uses the User Name field and the APPLICATION_DEVELOPER name.

*Figure 14-42  Completing the Select and Add Dialog*

![Select and Add: People](image)

7. Highlight the name you want to use.

8. Click **OK**. The information is entered on the Proxies pane, as shown here.

*Figure 14-43  Proxy Name Entered In Proxies Pane*

![Proxies](image)

9. Enter the necessary dates in the Start Date and End Date fields and click **Save**.

Now Impersonatee has granted proxy access to APPLICATION_DEVELOPER. APPLICATION_DEVELOPER can impersonate Impersonatee and perform transactions on his or her behalf.

10. Open the Settings and Actions menu and click **Sign Out**.

11. Sign back in as APPLICATION_DEVELOPER using the password for that user ID.

12. Click **Switch To**. The menu shows the names of the logged-in user and the users he or she can impersonate.

*Figure 14-44  Switch To Menu*

![Settings and Actions](image)

13. Select the name of the Impersonatee. You will be asked to confirm the switch.
The name of the impersonatee now is displayed with an icon indicating that the user is being impersonated.
This chapter discusses how to implement search functions in the UI Shell page template that is used to build web pages.

This chapter includes the following sections:

- Adding Attachments Support to Global Search
- Implementing Recent Items
- Implementing the Watchlist
- Implementing Group Spaces
- Implementing Activity Streams and Business Events
- Implementing the Search Results UI
- Implementing Auto-suggest
- Attachments in Global Search
- Supporting Global Search Alternate Words in the Cloud
- Implementing List Search

For more information about the features, see:

- Getting Started with Your Web Interface
- Implementing the UI Shell
- Implementing Additional Functions in the UI Shell
- Implementing UIs in JDeveloper with Application Tables, Trees and Tree Tables
- Implementing Applications Panels, Master-Detail, Hover, and Dialog Details

15.1 Adding Attachments Support to Global Search

This section assumes that you already have performed the basic attachments setup tasks on your searchable view object, just like you would any other view object.

Oracle Fusion applications has support for attachments using functionality delivered by Oracle Enterprise Crawl and Search Framework (ECSF). This functionality is integrated at crawl time of the Search view object, whereby ECSF will look at the crawled Search view object for attachment child view objects. If there are attachments defined this way, it will use Oracle Fusion Middleware Extensions for Applications Attachments APIs to retrieve the contents of the attachment, and they are added to the crawled document. All attachments found this way are added, and become a blob of
data in the indexed document. All references to the file name, file type, and so on, are lost. Refer to the ECSF class oracle.ecsf.fusion.FusionAttachmentImpl for details.

Note that existing ECSF implementation will only crawl attachments attached to the Search view object, not the attachments of child objects. For example, a Purchase Order with attachments will have the attachments crawled, but if the Purchase Order has Purchase Order lines, which also have attachments, these will not be crawled.

Searching attachments includes these features:

- **Keyword Search**: Users can perform a keyword search through several different entities within Oracle Fusion applications.
- **Search results drilldown**: Users can drill down on a specific record or documents included in the search results UI.
- **Entity keyword search**: Users can perform keyword search on a specific entity.
- **Keyword search configuration**: Administrators can configure the behavior of the keyword search by specifying what entities and fields will be used during the search process.
- **Attachments Search Security**: Attachments inherit security from the parent object to which the attachment is attached. This parent object security should be taken in consideration when defining attachments on searchable view objects and exposing them to the ECSF crawl.

**Overview of Attached Documents Search**

In a similar way to the existing ECSF crawl functionality, the Applications Core PreIndexProcessor can detect Attached documents at crawl time. It then stores in the indexed document this additional metadata about those attached documents to use when rendering the search result:

- Document Id
- Document Name
- Document Type

At search time, this data is used to show secured links in the existing search result so the user can click them and open the attachment.

Advantages of the Applications Core Attachments search functionality include:

- Minimal increase in data volumes or crawl time.
- Attachments are immediately available from the search results screen and can be opened in the same way as they can on the attachments UI.
- Attachments can be searched by file name and type.

**15.1.1 How ECSF Crawl Operates**

The ECSF Crawl of attachments does not handle categories or data security. In fact, due to the way it breaks the view links, it breaks the data security built into the view link and attachment code that uses it. No data security clause is added to the attachment query.
Background on how ECSF crawls master/detail relations

When ECSF crawls a Search view object that has detail relations, it does not use the default BC4J view links to generate the object tree of view objects. It breaks the view links and runs the master and detail queries separately, then joins the data using knowledge of the foreign keys in the view link. (This applies also to grandchildren and has nothing to do with attachments specifically; however, because an attachment is implemented as a child object, it is handled this way.)

For example, in crawling a simple Search view object with attachments, the basic algorithm is:

For each batch of Search view object row keys (typically 100)
  Query the Search view object for batch (1),
  Obtain key values of parent end of view link,
  Query child (attachment view object) with IN clause with parent PK values (2),
  For each record, match the parent (1) record with the child (2) based on the
key values defined for view link.
End For;

This shows an example of the attachment query for (2). PK1_VALUE and ENTITY_NAME are the attributes used in the view link in this example, and the values 1, 2, 3 and 'GS_PRODUCTS' come from the master Search view object.

```
SELECT * FROM (SELECT FndAttachedDocumentsEO.ATTACHED_DOCUMENT_ID,
    FndAttachedDocumentsEO.DOCUMENT_ID,
    FndAttachedDocumentsEO.LAST_UPDATE_DATE,
    FndAttachedDocumentsEO.LAST_UPDATED_BY,
    FndAttachedDocumentsEO.ENTITY_NAME,
    FndAttachedDocumentsEO.PK1_VALUE,
    FndAttachedDocumentsEO.PK2_VALUE,
    FndAttachedDocumentsEO.PK3_VALUE,
    FndAttachedDocumentsEO.PK4_VALUE,
    FndAttachedDocumentsEO.PK5_VALUE,
    FndAttachedDocumentsEO.DOCUMENT_ID AS DOCUMENT_ID1,
    FndDocumentsEO.DATATYPE_CODE,
    FndDocumentsEO.DESCRIPTION,
    FndDocumentsEO.FILE_NAME,
    FndDocumentsEO.USAGE_TYPE,
    FndDocumentsEO.DM_NODE,
    FndDocumentsEO.DM_FOLDER_PATH,
    FndDocumentsEO.DM_TYPE,
    FndDocumentsEO.DM_DOCUMENT_ID,
    FndDocumentsEO.DM_VERSION_NUMBER,
    FndDocumentsEO.URL,
    FndDocumentsEO.TITLE,
    decode(datatype_code, 'SHORT_TEXT', nvl(FndDocumentsEO.title, 'Undefined'),
    'FILE', nvl(FndDocumentsEO.title, FndDocumentsEO.file_name), 'WEB_PAGE',
    nvl(FndDocumentsEO.title, FndDocumentsEO.url)) attachment,
    FndAttachedDocumentsEO.ENTITY_ATTRIBUTES,
    FndDocumentsEO.DM_REPOSITORY,
    FndDocumentEntitiesEO.DOCUMENT_ENTITY_ID,
    FndAttachedDocumentsEO.CATEGORY_NAME,
    FndDocumentsEO.STATUS,
    FndDocumentsEO.DOWNLOAD_STATUS,
    FndDocumentCategoriesEO.CATEGORY_ID,
    FndDocumentCategoriesEO.USER_NAME,
    FndDocumentCategoriesEO.MODULE_ID,
    FndDocumentsEO.URI,
    FndDocumentsEO.ENT_APP_SHORT_NAME,
```

Implement Search Functions in the UI Shell
What this means is that all attachments directly attached to the Search view object are crawled, regardless of their category or the existence of security.

As the attachment is effectively inlined in the searchable document (this is how any attachment in SES currently works), a user with access to the Search view object can search on content from a secured attachment, even if the user has no data security rights on the attachment.

There is a fundamental mismatch in the security of the Search view object and secured attachments due to the “public user” nature of the crawl. The security of the attachment is effectively promoted to the functional security of the Search view object.

These secure attachments should not be crawled if they cannot be independently secured.

Due to this breaking of the View Link, if more than one child attachment is present, the attachments will be doubled up. This is because the View Link does not capture the attachment category (it is a custom attribute on the View Link).
15.1.2 How the Global Search Results UI Shows Existing Attachments

This section explains the design and behavior of how the results UI shows attachments.

The design:

• Shows attachments in the results UI using the same descriptions, tooltips, and so on, that the attachments component does.

• Allows the user to click an attachment in the results UI and have the attachment open.

• Shows keyword bolding in the attachment name (if any).

• Allows developers to promote grandchildren attachments if they want. So, for example, Purchase Order line attachments are shown with the Purchase Order result.

• Works across attachment categories.

• Suppresses secured attachments.

• Picks up Search view objects with changes attachments and flag for recrawling (but not grandchildren attachments).

• Works with the existing tag crawl functionality.

15.1.3 How Crawling Is Done

Crawling is done using the FndSearchPlugin that can be referenced by developers, either directly or as a base class to existing plugins.

The TagSearchPlugin is deprecated and stubbed with the removal of Webcenter tags. Developers should switch to FndSearchPlugin.
There are two arguments to determine how attachments are crawled:

- **ATTACHMENT_VIEW_LINK_ACCESSOR_PATHS** - A comma-separated list of accessor paths to Attachment view objects.

- **ATTACHMENT_INCREMENTAL_CRAWL_ENTITY_NAME** - The name of the attachment entity used in incremental crawls to reverse lookup changed attachments.

**Attribute ATTACHMENT_VIEW_LINK_ACCESSOR_PATHS**

This *required* attribute defines the attachments that will be considered as part of the crawl. Without it, the default attachment behavior of ECSF will be unchanged. That is, all direct attachments of the Search view object will be crawled, but nothing will be shown in the Search UI.

It is possible to have multiple attachment view links from a master view object, as well as attachments stored against detail view objects in a master-detail relation.

An accessor path is a dot-separated set of accessors to an Attachment view object. For example, `GsProductAttachmentsAccessor` is the view link accessor to the AttachmentVO directly under the Search view object.

`GsProductDetails.GsProductDetailsAttachmentsAccessor` is for an attachment under a detail-level view object.

Any number of these paths can be specified, although it is expected to be small (likely only 1).

As an example, accessors in the Master view object for this would be:
Accessors in the Detail view object for this would be:

When creating master-detail relations like this, do not include the accessor in the detail view object. If you do, ECSF will try to follow the links from Master to Detail to Master to Detail until it reaches the maximum depth of 7 levels.

In this case, you will see a crawl error similar to:

**Attribute ATTACHMENT_INCREMENTAL_CRAWL_ENTITY_NAME**

This attribute defines the attachment entity name that will be used when determining which Search view object records need to be flagged for re-crawl because they have had changes to their attachments. Because adding an attachment does not change the Last_update_date of the parent view object row, there needs to be a reverse lookup mechanism to force these additional rows onto the incremental crawl queue.

Note that only newly-added attachments and updated attachments (including the title) will be found for recrawl. There is no mechanism to detect deleted attachments.

This attribute is not required. If it is not supplied, attachment additions to a view object row that is otherwise unchanged (and hence on last_update_date change) will go un-noticed.

**Internal Stored Attribute FND_ATTACHMENT_METADATA**

Attachments found during the crawl will be recorded in the Search view object in the predefined stored string attribute FND_ATTACHMENT_METADATA. This attribute needs to be defined by the developer in the Search view object definition and flagged as a stored attribute. It should have no data in it; it will be over-written by the Applications Core preIndexProcessor at crawl time. An example of defining an attribute is shown here using a transient attribute.
What is stored in this attribute is an XML structure for all the attachments stored against a searchable object. As an example, here are four attachments stored against a row:

```xml
<Attachments>
  <Attachment documentId="300100024947778" datatypeCode="FILE" title="Latte.doc" fileName="Latte.doc" />
  <Attachment documentId="300100024947776" datatypeCode="FILE" title="UIShell_dtpreview.png" fileName="UIShell_dtpreview.png" />
  <Attachment documentId="300100024881786" datatypeCode="WEB_PAGE" title="oracle" description="Oracle homepage" fileName="http://oracle.com" />
  <Attachment documentId="300100024881784" datatypeCode="TEXT" title="a quick note" description="a quick note" fileName="a quick note" />
</Attachments>
```

Having these attributes, which are defined in this table, stored in the Search view object means they are searchable by default, and the UI can use the values to quickly decode the values without costly database look-ups when rendering the result.
Table 15-1 Attributes Definitions

<table>
<thead>
<tr>
<th>datatype Code</th>
<th>documentId</th>
<th>Title</th>
<th>Description</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>documentId</td>
<td>nvl(title, fileName)</td>
<td>description</td>
<td>fileName</td>
</tr>
<tr>
<td>WEB_PAGE</td>
<td>documentId</td>
<td>nvl(title, URL)</td>
<td>description</td>
<td>URL</td>
</tr>
<tr>
<td>TEXT</td>
<td>documentId</td>
<td>nvl(title, fileName)</td>
<td>description</td>
<td>fileName</td>
</tr>
<tr>
<td>TMURL</td>
<td>Not supported and there will be no attributes stored for it in SES.</td>
<td>nvl(title, URI)</td>
<td>description</td>
<td>URI (not URL)</td>
</tr>
<tr>
<td>FOLDER</td>
<td>Not supported and there will be no attributes stored for it in SES.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

15.1.3.1 Mapping AttachmentVO Attributes into Search Metadata

Note that the XML is supplied without a schema; it is an internal part of applcore search and may change at any time without notice.

Attachments found at detail levels are pulled up to the parent level. If this is not required, do not include an accessor path to this level so these attachments will not be crawled.

Attachments in all categories are included. However, if that document entity has security enabled (no matter what the data security rules are), the attachments assigned to that document entity are removed from the indexed document and therefore not crawled. Their details will not be in the example XML.

The following types of attachments will be available as links:

- **FILE**
  An attached file. This is the primary use case.

- **WEB_PAGE**
  A URL, such as http://www.companyname.com. The link will be placed verbatim in the results UI as an af:goLink. Note that ECSF does not crawl Web page attachments.

- **TEXT**
  Text typed by the user. This is implemented by attachments internally as a hidden file, so is practically the same as FILE.

The following types of attachments will not be available as links:

- **FOLDER**
  A reference to a folder in Oracle WebCenter Content. Requires use of a WebCenter Content folder taskflow.

- **TMURI**
Topology Manager URI. This is a URI offset to a web application defined in topology. This is a minor use case used in migration and is not supported by search.

15.1.4 How Attachments Are Shown

Attachments will be shown on a line below the Other Actions in the Search Results. That is the lowest part of a result item.

The text of the link will be the Attachment title, which generally defaults to the file name in the Attachments UI, but may be over-ridden by the user when the attachment is uploaded.

The tool tip of the attachment will be the attachment description entered by the user. If this is not present, the title value will be used.

Attachments will be sorted and displayed in the results alphabetically.

All information is available in the stored attribute of the Search view object to render attachment links in the UI with no additional lookup work, and therefore no significant performance overhead.

Figure 15-3 Attachments Displayed in the Search Results

15.1.5 How Attachments Are Opened

Clicking an attachment will open the content for the user. The browser settings will determine how this is done and should be in line with what the Attachment component does.
For example, clicking a JPG attachment normally will open it directly in a browser tab, while clicking a .doc file will likely open it in Microsoft Word; this depends on the individual web browser configuration.

The different attachment types have different requirements to be able to download the attachment; some can be directly clicked; others need intermediate lookups done in the middle tier.

In addition, there is a security implication for attachments stored in WebCenter Content. The "real" url to the WebCenter Content attachment is long and needs to be determined within a few minutes of being used, because it has an implicit timeout in it. This means it cannot be pre-determined and stored in the Oracle Secure Enterprise Search Indexed Document. Similarly, on rendering of the results, it is not possible to look up the link in an efficient manner and show the user an af:goLink to the Attachment for all attachments in the search result. (The user may walk away from the screen and come back and click the link.)

The link that the user clicks needs to do the work of determining the "real link" at the point of click.

15.1.6 How to Implement Attachments Search

There are two cases to consider:

- If the Search view object has no plugin currently configured, simply use the Applications Core plugin.
- If the Search view object has a plugin already, set up the existing plugin to inherit the Applications Core plugin.

15.1.6.1 Using the Applications Core Plugin

In the ECSF Search view object design time, add a Search Plugin to the Applications Core class oracle.apps.fnd.applcore.search.plugin.FndSearchPlugin.
Configure attachments that will be crawled

Add a parameter with a comma-separated list of accessor paths with the key
ATTACHMENT_VIEW_LINK_ACCESSOR_PATHS.

Configure Search view object incremental crawl detection of Search view objects with new attachments

Optionally, add an attribute with the key
INCREMENTAL_CRAWL_ATTACHMENT_ENTITY_NAME and a value of the Attachment Entity Name (the value you enter in the Attachment View Link wizard, such as GS_PRODUCTS).

This will add Search view objects to the crawl list when new attachments are added. This is useful because the Search view object last_update_date value is not updated when an attachment is added, and the document will not be detected as needing a re-crawl. This process will flag the individual Search view object row to be recrawled. Note that it is not possible to detect changed attachments this way at the detail level (for example a purchase order line's attachments), because the reverse lookup would obtain the primary key of the line item and not the Search view object.

15.1.6.2 Enhancing an Existing Plugin

Developers that have an existing plugin should extend the Applications Core plugin and call super() on the methods that implement attachment functionality. Use the same parameters to the plugin as already described.

```java
public class AppsSearchPlugin
extends FndSearchPlugin
implements PreIndexProcessor,
ChangeListener
{
/**
 * Hook into the pre index process of the crawl so we can annotate the Search view object
 * with apps specific information such as tags and attachment metadata.
 * @param ctx Search Context.
 * @param docs Documents to iterate and find tag children for.
 */
public void preIndexProcess(SearchContext ctx,
List<IndexableDocument> docs)
{
  super.preIndexProcess(ctx, docs);
  // custom apps functionality ...
}
/**
 * Get the change list for crawling.
 * @param ctx Search Context.
 * @param changeType the type of the change requested. Must be one of following
 * <ul>
 * <li>ChangeListener.UPDATE</li>
 * <li>ChangeListener.INSERT</li>
 * <li>ChangeListener.DELETE</li>
 * </ul>
 * @return an iterator of a list of primaryKeys. Returns an empty iterator if nothing changed.
 */
public Iterator getChangeList(SearchContext ctx, String changeType)
{
  Iterator ret = super.getChangeList(ctx, changeType);
  // custom apps functionality ...
}
```

Adding Attachments Support to Global Search
15.1.6.3 Adding the Transient Stored Attribute FND_ATTACHMENT_METADATA

Add a transient string attribute to the Search view object named FND_ATTACHMENT_METADATA to store Applications Core attribute metadata. Flag this attribute as stored in the Search view object definition.

15.2 Implementing Recent Items

Recent Items tracks a list of the last 20 task flows visited by a user. The Recent Items list is persistent across user sessions and a task can be restarted from the Recent Items list. The feature is automatically turned on and will be available automatically in pages using the UI Shell template. Security must be disabled to turn Recent items off.

Before You Begin

For the Recent Items feature to work, you must configure the user session and ADF Security. See Implementing Application User Sessions. Without security enabled, recent items will not be captured because the data is recorded for each authenticated user.

15.2.1 How to Choose Labels for Task Flows

Recent Items records the task flow labels for a task flow that has started. Therefore, you must carefully choose the labels for task flows, and must provide task flow labels for all task flows, even if they are meant to be used in no-tab mode (see Supporting No-Tab Work Areas).

15.2.2 How to Call Sub Flows

The openMainTask method is used to open a new task in the main area of Oracle Fusion web applications that use UI Shell templates. Besides opening a new tab, openMainTask also pushes a new task flow history object onto a stack, which is used to keep track of all task flows that have been opened. The task flow ID and its associated parameter values are encapsulated in the task flow history object.

Having this information, the call to closeMainTask pops the stack to get the last task flow ID and its parameter values that were displayed, and reinitializes the main area with that task flow and parameter information.

When a task flow is called from the local area task flow using task flow call activity, it is called a sub flow. By default, sub flows will not be recorded on the stack as described. Two new APIs are exposed in FndUIShellController data control for registering sub flows: openSubTask and closeSubTask.

15.2.2.1 Sub Flow Registration APIs

Use the openSubFlow and closeSubFlow APIs to record sub flows to Recent Items. Whenever an ADF Controller task flow call takes place, no notification is raised to Applications Core or UI Shell. So, unlike starting a task from a tasks list, you need to explicitly notify the UI Shell for sub flow calls.

When the openSubTask API is called before a subflow is started, the sub flow ID and its parameter values are pushed onto the stack. Applications Core also notifies the Recent Items implementation with recorded task flow information. This essentially
makes a sub flow able to be bookmarked by Recent Items, and can be started directly from the selection of menu items on Recent Items.

Note that registering sub flows to Recent Items is optional.

**Implementation**

This API is exposed as the data control methods
FndUIShellController.openSubTask and
FndUIShellController.closeSubTask that can be dragged and dropped to page fragments to create links to notify the UI Shell. The FndUIShellController data control is automatically available to all Oracle Fusion applications that reference Applications Core libraries.

Example 15-1 shows the signature and Javadoc of the method.

All the parameters required to be passed in openSubTask are exactly same as used by the Navigate API. For more information, see Introducing the Navigate API.

**Example 15-1  Recent Items API**

```java
/**
 * Notify UIS hell to record a given sub flow on the stack and
 * also notify Recent Items to include it on the list.
 *
 * @param taskFlowId Task flow to open
 * @param parametersList Parameters list for the task flow.
 *                    This is a semicolon delimited String
 *                    of name-value pairs. For example,
 *                    "param1=value1;param2=value2".
 *
 * @param label Label for the task flow.
 * @param keyList Key list to locate the task flow instance.
 *                This is a semicolon delimited keys or key-value pairs.
 *                For example, "key1;key2=value2". If only the key is specified,
 *                the value is picked up from parametersList with the same
 *                name as the key.
 *
 * @param taskParametersList Parameters list to pass in to the task flow to open
 *                            in the target workspace. This is a semicolon delimited String
 *                            of name value pairs. For example,
 *                            "param1=value1;param2=value2".
 *
 * @param methodParameters This can be used for passing
 *                            Java object into the task flow that is specified
 *                            in taskFlowId parameter. Use <code>setCustomObject() API</code>
 *                            in FndMethodParameters for setting the java object.
 *
 * @return For internal Contextual Event processing
 */

public FndMethodParameters openSubTask(String taskFlowId,
String parametersList,
String label,
String keyList,
String taskParametersList,
String viewId,
String webApp,
FndMethodParameters methodParameters)

/**
 * Closes the currently-focused sub-task and the focus moves to the
 * task from which this sub-task was started.
 */
```
public FndMethodParameters closeSubTask(FndMethodParameters methodParameters)

### 15.2.2.2 openSubTask API Labels

The openSubTask API accepts the same set of parameters as used by the Navigate API. If no label is specified in the openSubTask API, Recent Items will register it with the name of the parent task flow’s label. You should set a different label based on the business use case in the openSubTask API. Failing to do so will register this flow with the same label as the parent task flow’s label and, therefore, will make it impossible to distinguish between the parent flow and the sub flow entry in the Recent Items list.

### 15.2.2.3 Starting from Recent Items

Whatever task flow details are registered while invoking the openSubTask API will be used by Recent Items to start it. Recent Items takes care of starting the task flow in the right work area and web application. You do not need to do anything for that. Because the openSubTask API supports parameters, you can pass some requirement-specific values to it while registering the task flow to Recent Items. On starting, those passed values are available in the pageFlowScope. You can analyze these values and make decisions, such as if you need to first initialize the parent flow, or if you need to set Visible to False on some of the actions on the page.

### 15.2.3 How to Enable a Sub Flow to Be Bookmarked in Recent Items

To record sub flows into the Recent Items list, applications need to call the openSubTask API right before sub flows are started. openSubTask takes parameters similar to the Navigate API. One of these is task flow ID. For this, you need to specify the parent flow’s ID (or main task’s ID). In other words, sub flows need to be executed by using the parent flow, even though they are started from the Recent Items menu.

If your sub flow does not need to be bookmarked by Recent Items, you do not need to change anything. Otherwise, you need to modify your parent flow and sub flow as described in this section. After the changes, sub flows can be started in two ways:

- From original flows
- From Recent Items menu items using recorded information

Both will start the execution in the parent flow. Because the sub flow must be piggybacked on the parent flow when it is started from the Recent Items menu, you need to change the parent flow following these directions:

- Add a new router activity at the beginning of the parent flow. Based on a test condition, it will route the control to either the original parent flow or the task flow call activity (that is, the sub flow).
- Add an optional method call activity to initialize the sub flow before it is started from the Recent Items menu. Product teams can code the method in such a way that it can navigate to the sub flow after initializing the parent state. This allows product teams to render the contextual area, navigating back to the parent flow from the sub flow, and any other customizations.
• Bind `openSubTask` to the command component (such as a link or button) which causes the flow to navigate to the task flow call activity in the original parent flow. The `openSubTask` API registers the parent flow details and input parameters to the sub flow (to be started as a sub flow later) to the Applications Core task flow history stack.

Usually, you do not need to modify your sub flow for this task. However, you can consolidate the initialization steps from two execution paths in this way:

• Remove initialization parts from both paths in the parent flow. Instead, set input parameters in both paths only.
• Modify the sub flow to take input parameters.
• Add a new method call (such as `initsubflow`) at the beginning of the sub flow to initialize states in the parent flow (for example, parent table) so that the sub flow can be started in the appropriate context.

Note that the design pattern also requires the application to be able to navigate back to the parent flow from the sub flow. The initialization code should take this into consideration, such as by setting up states to allow the sub flow to navigate back.

In this example, you will use an Employee sample implementation to demonstrate the details of this design pattern.

**Sub Flow Sample Application**

As shown in Figure 15-5, users select *Subflow Design Pattern* from the task list. They then specify some criteria to search for a specific employee or employees. From the list, they can choose the employee for whom they want to show the details.

*Figure 15-5  Example List of Employees*

The Ename column in the search result table is a link that can be used to navigate to the employee detail page of a specific employee. When this link is clicked, a sub flow (or nested bounded task flow) is called to display the Employee Complete Detail page, as shown in Figure 15-6.
15.2.3.1 Implementing the Sub Flow Design Pattern

The parent task flow named `ToParentSFFlow` is shown in Figure 15-7.

The router activity `decideFlow` decides whether the control task flow should go to the original parent task flow path (`initParent`) or to the sub task flow path (`toChild`). The condition is defined as:

```xml
<router id="decideFlow">
  <case>
    <expression>#{pageFlowScope.Empno == null}</expression>
    <outcome id="__9">initParent</outcome>
  </case>

  <case>
    <expression>#{pageFlowScope.Empno != null}</expression>
    <outcome id="__10">toChild</outcome>
  </case>

  <default-outcome>initParent</default-outcome>
</router>
```
The test checks whether the Empno variable in the parent task flow's pageFlowScope is null. #{pageFlowScope.Empno} is set using its input parameter Empno when the parent task flow is called. The input parameters on the parent task flow (that is, ToParentSFFlow) are defined as:

```
<input-parameter-definition>
  <name>Empno</name>
  <value>#{pageFlowScope.Empno}</value>
  <class>java.lang.String</class>
</input-parameter-definition>
```

When the parent task flow is started from the task list, the Empno parameter is not set (that is, it is not defined in the application menu's itemNode). Therefore, the parameter is null and the router will route it to the initParent path.

When the sub task flow is recorded through the openSubTask API, the Empno parameter is set on parametersList as:

```
<methodAction id="openSubTask" RequiresUpdateModel="true"
  Action="invokeMethod" MethodName="openSubTask"
  IsViewObjectMethod="false" DataControl="FndUIShellController"
  InstanceName="FndUIShellController.dataProvider"
  ReturnName="FndUIShellController.methodResults.openSubTask_FndUIShellController_dataProvider.openSubTask_result">
  <NamedData NDName="taskFlowId" NDType="java.lang.String"
    NDValue="/WEB-INF/oracle/apps/xteam/demo/ui/flow/ToParentSFContainerFlow.xml#ToParentSFContainerFlow"/>
  <NamedData NDName="parametersList" NDType="java.lang.String"
    NDValue="Empno=#{row.Empno}"/>
</methodAction>
```

You also set up:

- `taskFlowId` to be the parent task flow's, not the sub task flow's
- `label` to be the sub task flow's

When users click the link (the Ename) to which the openSubTask method is bound, openSubTask will be called. This link component is defined as:

```
<af:column sortProperty="Ename" sortable="false"
  headerText="#{bindings.ComplexSFEmpVO.hints.Ename.label}"
  id="resId1c2">
  <af:commandLink id="ot3" text="#{row.Ename}"
    actionListener="#{bindings.openSubTask.execute}"
    disabled="#{bindings.openSubTask.enabled}"
    action="toChild">
  </af:commandLink>
</af:column>
```
Note that when the link is clicked:

- **actionListener** and the **action** specified on the link are executed, in that order.

- **openSubTask** is called only from the original parent task flow path (that is, **initParent**), not from the sub task flow path (that is, **toChild**).

The **EmployeeDetails** activity in Figure 15-7 is a Task Flow Call activity that invokes the **ToChildSFFlow** sub task flow. Before the sub task flow is executed, add initialization steps. These initialization steps could include, but are not limited to:

- Set up parent states. For this example, set the selected employee's row to be the current row.

- Set up the Contextual Area state.

- Set up states to allow the sub task flow to navigate back to the parent task flow.

There are two approaches to setting up the initialization steps:

- In the parent task flow

- In the sub task flow

For the first approach, you can add logic to initialize both paths before the task flow call activity in the parent task flow. For the second approach, you initialize states in the sub task flow by using input parameters of the sub task flow. For example, the sub task flow will take an input parameter named **Empno**. In effect, the second approach just postpones the initialization to the sub task flow.

The definition of input parameters in the task flow call activity is:

```xml
<task-flow-call id="EmployeeDetails">
  <task-flow-reference>
    <document>/WEB-INF/oracle/apps/xteam/demo/ui/flow/ToChildSFFlow.xml</document>
    <id>ToChildSFFlow</id>
  </task-flow-reference>
  <input-parameter>
    <name>Empno</name>
    <value>#{pageFlowScope.Empno}</value>
  </input-parameter>
</task-flow-call>
```

Note that this means that the calling task flow needs to store the value of **Empno** in \#{pageFlowScope.Empno}. For example, from the original parent task flow path, it is set to be \#{row.Empno} using the **setActionListener** tag. For the sub task flow path, it is set using the parent task flow's input parameter **Empno**. On the sub task flow, you need to specify its input parameters as:

```xml
<task-flow-definition id="ToChildSFFlow">
  <default-activity>TochildSFPF</default-activity>
  <input-parameter-definition>
    <name>Empno</name>
    <value>#{pageFlowScope.Empno}</value>
    <class>java.lang.String</class>
  </input-parameter-definition>
  ...
</task-flow-definition>
```
Note that the name of the input parameter (Empno) must be the same as the parameter name defined on the task flow call activity. When the parameter is available, Oracle ADF will place it in #{pageFlowScope.Empno} to be used within the sub task flow. However, this pageFlowScope is different from the one defined in the task flow call activity because they have a different owning task flow (that is, parent task flow versus sub task flow).

The definition of the sub task flow is shown in Figure 15-8:

**Figure 15-8   Example Sub Task Flow Definition**

In the sample implementation, you implemented the initialization step in the sub task flow. The Empno variable is passed as a parameter to the sub task flow and used to initialize the parent state. When the sub task flow is started, the default view activity (TochildSFPF) is displayed. Before it renders, the initPage method on the ChildSFBean will be executed. The page definition of the default page is defined as:

```xml
<pageDefinition xmlns="http://xmlns.oracle.com/adfm/uimodel">
  <parameters/>
  <executables>
    ...
    <invokeAction id="initPageId" Binds="initPage" Refresh="always"/>
  </executables>
  <bindings>
    ...
    <methodAction id="initPage" InstanceName="ChildSFBean.dataProvider"
                  DataControl="ChildSFBean" RequiresUpdateModel="true"
                  Action="invokeMethod" MethodName="initPage"
                  IsViewObjectMethod="false"
                  ReturnName="ChildSFBean.methodResults.initPage_ChildSFBean_dataProvider_initPage_result"/>
    ...
  </bindings>
</pageDefinition>
```

The initPage method is specified in the executables tag and will be invoked when the page is refreshed. The initPage method itself is defined as:

```java
public void initPage() {
    FacesContext facesContext = FacesContext.getCurrentInstance();
    ExpressionFactory exp = facesContext.getApplication().getExpressionFactory();
    DCBindingContainer bindingContainer =
        (DCBindingContainer)exp.createValueExpression(
```
facesContext.getELContext(),"#{bindings}",DCBindingContainer.class).getValue(facesContext.getELContext());
ApplicationModule am = bindingContainer.getDataControl().getApplicationModule();

ViewObject vo = am.findViewObject("ComplexSFEmpVO");
vo.executeQuery();

Map map = AdfFacesContext.getCurrentInstance().getPageFlowScope();
if(map !=null){
    Object empObj = map.get("Empno");
    if(empObj instanceof Integer){
        Integer empno = (Integer)map.get("Empno"); // new Integer(empnoStr);
        Object[] obj = {empno};
        Key key = new Key(obj);
        Row row = vo.getRow(key);
        vo.setCurrentRow(row);
    } else {
        String empnoStr = (String)map.get("Empno");
        Integer empno = new Integer(empnoStr);
        Object[] obj = {empno};
        Key key = new Key(obj);
        Row row = vo.getRow(key);
        vo.setCurrentRow(row);
    }
}

The initPage method takes the input parameter Empno from
#{pageFlowScope.Empno} as a key to select a row and set it to be the current row
in the master Employee table.

15.2.4 How to Use Additional Capabilities of the Recent Items openSubTask API

The openSubTask API has additional capabilities. For example, consider an
employee search page in which you enter parameters such as department number and
manager ID, and search for the matching employee records. You can use the
openSubTask API to register a search page with search parameters. The next time the
user can see the search results by just starting it from the Recent Items menu. This is
similar to using parametersList to specify search parameters while registering the
search task flow. While starting, additional programming can be done to retrieve the
search parameters and execute the query with the parameter values.

Favorites

As soon as tasks are recorded on the Recent Items list, they are eligible for Favorites.
The Favorites menu is implemented on top of Recent Items. Any current task on the
Recent Items list can be bookmarked and placed in the Favorites folders. Currently,
only a one-level folder is supported. Similar to Recent Items, tasks on the Favorites list
can be started directly from the list. So, the description in this section for Recent Items
applies also to the Favorites implementation. For example, sub flows based on the
design pattern described in this section can be registered on the Favorites list as well
as the Recent Items list.
15.2.5 How to Implement Data Security for Recent Items and Favorites

Data security controls access to the data displayed on the target. When data security is enforced on the target page, any changes to the access that the user has is properly reflected on the data.

However, there are use cases in Oracle Fusion Applications where the Recent Items and Favorites features may cause security issues. In some cases there is a master/detail relationship between objects in which only the parent object has data security implemented. The child object has no data security enforced and is expected to inherit security from the parent. In such cases, the user can only navigate to the child task flow through the parent task flow. But because the child task flow can be added as a Recent Item or Favorite, users can navigate directly to the child through Recent Items, whether they still have access to the parent.

Data security ensures that user access is properly enforced on all flows that can be accessed from Recent Items or Favorites.

Implementation

Implementation consists of ensuring that security is enforced consistently in all scenarios and that authorization exceptions are handled by the standard central method.

Security Enforcement

For function security, no additional implementation is required. The recommendations for data security are:

- Data security should be applied on an appropriate level. For example, in a master/detail relationship, security definitions should be created on both parent and child objects and applied on a proper level.

- If the child record inherits the security from its parent, then the parent’s security must be programmatically enforced on a sub flow. This should be done by adding a default method activity on the task flow and calling the data security API to check that the user has access to the parent. If the user does not have proper access, an authorization exception is generated.

  In addition, you should programmatically enforce the parent data security where applicable. For instance, if a view object takes the parent primary key as a parameter, before executing the query the code should check that the user has access to the parent.

15.2.6 Known Issues

- Recent Items are persistent across user sessions.

- You may see a null pointer exception when all the task flow parameters are not supplied values.

15.3 Implementing the Watchlist

The Watchlist is a portlet or menu, accessible to Oracle Fusion Applications users, that provides a summary of items that a user wants to track. The Watchlist includes seeded items (items that are provided out of the box) categorized by functional areas, and items created by the user. Technically, the Watchlist presents a list of pre-queried searches (saved searches or standard queries) of things the user wants to track. Each
item is composed of descriptive text followed by a count. Each item also is linked to a page in a work area where the individual items of interest are listed.

The Watchlist is available both as a dashboard region in the Welcome tab of the Home dashboard, and as a global menu. These are two views of the same content. The dashboard region is available to users as soon as they log in, while the global menu is accessible as they navigate through the suite.

The Watchlist will be refreshed to fetch new counts and items whenever the user navigates to the Home page. The Watchlist can refresh the entire list or individual categories as needed. Users can personalize the Watchlist to hide or show items.

Figure 15-9 shows an example of the Watchlist portlet and menu.

Figure 15-9   Example Watchlist Portlet and Menu

You have these high-level tasks:

- Code view objects with bind variables and default values for bind variables as needed for calculating the Watchlist count. These view objects will be executed at runtime by the Watchlist API to get the Watchlist count for the user.

- Code task flows to enable drill-down from the Watchlist UI.

- Seed FND deployments to specify host, port, and context root information for UI drilldown and service invocation.

- Seed FND standard lookups for each Watchlist category and item meaning.

- Seed information to tell the Watchlist what counts to track, how to display them to the user, which view objects to execute, and how to drill down to the work area and tasks.

- Import Watchlist JAR files as ADF libraries in service and UI projects.

- Set up a service interface method so that product code and Watchlist code can interact.
15.3.1 Watchlist Data Model Effects

Product teams will seed data into the ATK_WATCHLIST_CATEGORIES and ATK_WATCHLIST_SETUP tables. Rows in the ATK_WATCHLIST_ITEMS will be managed by Watchlist code, but you will query it for testing verification.

The only other data model effect will be in the creation of summary tables. These summary tables help with retrieving the count of Watchlist items with data security. See Summary Tables.

15.3.2 Watchlist Physical Data Model Entities

The Watchlist data model is supported by ATK. The tables are:

- **ATK_WATCHLIST_CATEGORIES**: Represents the functional categories in which each Watchlist item will fit. See Table 15-2.
- **ATK_WATCHLIST_SETUP**: Represents a type of count that a Watchlist item can track. The primary key is a Watchlist item code. See Table 15-3.

### Table 15-2  ATK_WATCHLIST_CATEGORIES

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>Required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>WATCHLISTCATEGORYCODE</td>
<td>VARCHAR2(100)</td>
<td>Yes</td>
<td>Primary Key - Unique code based on Product Code Prefix. Ensure that this code begins with &lt;PRODUCT SHORT CODE&gt;_, so that it does not overlap with others.</td>
</tr>
<tr>
<td>CATEGORY_LOOKUP_TYPE</td>
<td>VARCHAR2(30)</td>
<td>Yes</td>
<td>Reference to FND_STANDARD_LOOKUP_TYPES.LOOKUP_TYPE. Product teams will seed the lookup type with meaning for the category (VIEW_APPLICATION_ID = 0 and SET_ID = 0). The translated lookup type meaning is shown in the Watchlist UI for the category.</td>
</tr>
<tr>
<td>OWNING_MODULE_NAME</td>
<td>VARCHAR2(4000)</td>
<td>Yes</td>
<td>Reference to FND_APPL_TAXONOMY.MODULE_NAME for the owning product or module. This is used for seed data purposes.</td>
</tr>
<tr>
<td>OWNING_MODULE_ID</td>
<td>VARCHAR2(32)</td>
<td>Yes</td>
<td>Reference to FND_APPL_TAXONOMY.MODULE_ID for the owning product or module. This is used for seed data purposes.</td>
</tr>
<tr>
<td>OWNINGAPPLICATION_ID</td>
<td>NUMBER</td>
<td>Yes</td>
<td>Reference to FND_APPL_TAXONOMY.ALTERTATIVE_ID for the owning product or module. This is used for seed data purposes.</td>
</tr>
</tbody>
</table>
### Table 15-2  (Cont.) ATK_WATCHLIST_CATEGORIES

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>Required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFRESH_SERVICE_ENDPOINT_KEY</td>
<td>VARCHAR2(60)</td>
<td>Yes</td>
<td>The key to determine the host, port, context root, and so on to construct the URL for the service end point (wsdl location). This will be based on the Applications Core lookup API that will be used to determine the end point.</td>
</tr>
<tr>
<td>REFRESH_SERVICE_NAME</td>
<td>VARCHAR2(400)</td>
<td>Yes</td>
<td>The service that must be invoked for count calculation (for refreshing this category).</td>
</tr>
<tr>
<td>REFRESH_SERVICE_METHOD_NAME</td>
<td>VARCHAR2(400)</td>
<td>Yes</td>
<td>Obsolete. The hard-coded method name will be refreshWatchlistCategory. Product teams will create this hard-coded service method.</td>
</tr>
<tr>
<td>ENABLED</td>
<td>VARCHAR2(1)</td>
<td>Yes</td>
<td>Defaults to Y. Defines if this Watchlist category is enabled or active.</td>
</tr>
<tr>
<td>CREATED_BY</td>
<td>VARCHAR2(64)</td>
<td>Yes</td>
<td>Standard WHO Column</td>
</tr>
<tr>
<td>CREATION_DATE</td>
<td>TIMESTAMP</td>
<td>Yes</td>
<td>Standard WHO Column</td>
</tr>
<tr>
<td>LAST_UPDATED_BY</td>
<td>VARCHAR2(64)</td>
<td>Yes</td>
<td>Standard WHO Column</td>
</tr>
<tr>
<td>LAST_UPDATE_DATE</td>
<td>TIMESTAMP</td>
<td>Yes</td>
<td>Standard WHO Column</td>
</tr>
<tr>
<td>LAST_UPDATE_LOGIN</td>
<td>VARCHAR2(32)</td>
<td>Yes</td>
<td>Standard WHO Column</td>
</tr>
</tbody>
</table>

### Table 15-3  ATK_WATCHLIST_SETUP

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>Required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>WATCHLIST_ITEM_CODE</td>
<td>VARCHAR2(100)</td>
<td>Yes</td>
<td>Primary Key - Uses the Category Code Prefix</td>
</tr>
<tr>
<td>ITEM_LOOKUP_CODE</td>
<td>VARCHAR2(30)</td>
<td>Yes if</td>
<td>Reference to FND_LOOKUPS.LOOKUP_CODE Product teams will seed the lookup code with the meaning for the parent category lookup type. The translated lookup meaning is shown in the Watchlist UI with the count appended.</td>
</tr>
<tr>
<td>CATEGORY_CODE</td>
<td>VARCHAR2(30)</td>
<td>Yes</td>
<td>None</td>
</tr>
<tr>
<td>PRIVILEGE_BASED</td>
<td>VARCHAR2(1)</td>
<td>Yes</td>
<td>Identification of a Watchlist item that is created against a security action instead of a specific user.</td>
</tr>
<tr>
<td>OWNING_PRIVILEGE_NAME</td>
<td></td>
<td>Yes if</td>
<td>Defines if this item is created against a security action. Users that have this action will be able to view this item.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PRIVILEGE_BASED = Y</td>
<td></td>
</tr>
</tbody>
</table>

Implementing the Watchlist Search Functions in the UI Shell 15-25
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>Required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNCTION_PRIVILEGE_NAME</td>
<td>VARCHAR2(400)</td>
<td></td>
<td>Defines the region action for this item's drilldown work area. This is the page definition for the drilldown view or jspx that is part of your jazn-data.xml file. The user needs this permission policy to view the item in the Watchlist UI.</td>
</tr>
<tr>
<td>WATCHLIST_ITEM_TYPE</td>
<td>VARCHAR2(30)</td>
<td>Yes</td>
<td>Defines the Watchlist item type and maps to the lookup. Valid values are: SEEDED_QUERY (Seeded Query), SEEDED_SAVED_SEARCH (Seeded Saved Search), USER_SAVED_SEARCH (User-created Saved Search), HUMAN_TASK (Worklist item)</td>
</tr>
<tr>
<td>HUMAN_TASK_DEF_ID</td>
<td>VARCHAR2(200)</td>
<td>Yes if</td>
<td>Human Task Definition Identifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WATCHLIST_ITEM_TYPE = HUMAN_TASK</td>
<td></td>
</tr>
<tr>
<td>HUMAN_TASK_STATE</td>
<td>VARCHAR2(100)</td>
<td>Yes if</td>
<td>Human Task State Identifier.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WATCHLIST_ITEM_TYPE = HUMAN_TASK</td>
<td></td>
</tr>
<tr>
<td>REFRESH_AGE</td>
<td>NUMBER(9)</td>
<td>Yes if</td>
<td>Defines the time for a count in seconds. After this many seconds have passed since the last refresh time, the Watchlist UI will issue a count recalculation request</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WATCHLIST_ITEM_TYPE = HUMAN_TASK</td>
<td></td>
</tr>
<tr>
<td>VIEW_OBJECT</td>
<td>VARCHAR2(400)</td>
<td>Yes if</td>
<td>Complete path of the view object that must be executed for count calculation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WATCHLIST_ITEM_TYPE = HUMAN_TASK</td>
<td></td>
</tr>
<tr>
<td>SUMMARY_VIEW_ATTRIBUTE</td>
<td>VARCHAR2(400)</td>
<td>Yes if</td>
<td>Indicates the view object attribute name if using a summary view object for Watchlist count calculation. On execution, this view object returns only one row with the Watchlist item count.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WATCHLIST_ITEM_TYPE = HUMAN_TASK</td>
<td>and this is a summary view object</td>
</tr>
<tr>
<td>APPLICATION_MODULE</td>
<td>VARCHAR2(400)</td>
<td>Yes if</td>
<td>Complete path of the application module that contains the view object instance that must be executed for count calculation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WATCHLIST_ITEM_TYPE = HUMAN_TASK</td>
<td></td>
</tr>
<tr>
<td>AM_CONFIG_NAME</td>
<td>VARCHAR2(400)</td>
<td>Yes if</td>
<td>The application module configuration name for creating an instance of the application module from code. This is typically AMLOCAL.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WATCHLIST_ITEM_TYPE = HUMAN_TASK</td>
<td></td>
</tr>
</tbody>
</table>
Table 15-3 (Cont.) ATK_WATCHLIST_SETUP

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>Required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIEW_OBJECT_INSTANCE</td>
<td>VARCHAR2(400)</td>
<td>Yes if</td>
<td>The instance name for the view object in the application module.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WATCHLIST_ITEM_TYPE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PE != HUMAN_TASK</td>
<td></td>
</tr>
<tr>
<td>VIEW_CRITERIA_ID</td>
<td>VARCHAR2(400)</td>
<td>Yes if</td>
<td>The view criteria that must be applied when executing the view object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WATCHLIST_ITEM_TYPE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PE = SEEDED_SAVED_SEARCH</td>
<td></td>
</tr>
<tr>
<td>NAVIGATION_URL_KEY</td>
<td>VARCHAR2(60)</td>
<td>Yes</td>
<td>The key to determine the host, port, and context root to construct the URL for the UI drilldown for a Watchlist item. This will be based on the Applications Core lookup API that is used for UI navigation across Java EE applications.</td>
</tr>
<tr>
<td>VIEW_ID</td>
<td>VARCHAR2(400)</td>
<td>Yes</td>
<td>The view ID (as per the UI Shell menu) for the workarea or page that contains the task flow for this Watchlist item's drilldown from the Watchlist UI.</td>
</tr>
<tr>
<td>PAGE_PARAM_LIST_STRING</td>
<td>VARCHAR2(400)</td>
<td>No</td>
<td>Parameters list for the page. If the target workarea page accepts page parameters, this is a semicolon-delimited string of name-value pairs.</td>
</tr>
<tr>
<td>TASKFLOW_ID</td>
<td>VARCHAR2(400)</td>
<td>Yes</td>
<td>The task flow for this Watchlist item's drilldown from the Watchlist UI.</td>
</tr>
<tr>
<td>TF_KEY_LIST_STRING</td>
<td>VARCHAR2(400)</td>
<td>No</td>
<td>Key list to pass into the task flow to open in the target workspace. This is a semicolon-delimited list of keys or key-value pairs. For example, &quot;key1;key2=value2&quot;</td>
</tr>
<tr>
<td>TF_PARAMETER_STRING</td>
<td>VARCHAR2(400)</td>
<td>Yes if</td>
<td>Parameters list to pass in to the task flow to open in the target workspace. This is a semicolon-delimited string of name-value pairs. For example, &quot;param1=value1;param2=value2&quot; For a user-created saved search, the view criteria ID will be appended to this string (the string must end with &lt;paramName&gt;= for saved search).</td>
</tr>
<tr>
<td>TASK_TAB_LABEL</td>
<td>VARCHAR2(400)</td>
<td>Yes</td>
<td>Label for the task flow to open in the target workspace.</td>
</tr>
</tbody>
</table>
Table 15-3  (Cont.) ATK_WATCHLIST_SETUP

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Datatype</th>
<th>Required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled</td>
<td>VARCHAR2(1)</td>
<td>Yes</td>
<td>Defaults to Y. A definition that indicates if this Watchlist category is enabled or active.</td>
</tr>
<tr>
<td>created_by</td>
<td>VARCHAR2(64)</td>
<td>Yes</td>
<td>Standard WHO Column</td>
</tr>
<tr>
<td>creation_date</td>
<td>TIMESTAMP</td>
<td>Yes</td>
<td>Standard WHO Column</td>
</tr>
<tr>
<td>last_updated_by</td>
<td>VARCHAR2(64)</td>
<td>Yes</td>
<td>Standard WHO Column</td>
</tr>
<tr>
<td>last_update_date</td>
<td>TIMESTAMP</td>
<td>Yes</td>
<td>Standard WHO Column</td>
</tr>
<tr>
<td>last_update_login</td>
<td>VARCHAR2(32)</td>
<td>Yes</td>
<td>Standard WHO Column</td>
</tr>
</tbody>
</table>

15.3.3 Supported Watchlist Items

Supported Watchlist items are all *asynchronous* (that is, queries are executed on demand when the user requests a refresh or just before the Watchlist UI is shown). There are four types of asynchronous Watchlist items (watchlist_item_type):

- Seeded queries (SEEDED_QUERY)
- Seeded saved searches (SEEDED_SAVED_SEARCH)
- User-created saved searches (USER_SAVED_SEARCH)
- Human task-flow items (HUMAN_TASK)

15.3.3.1 Asynchronous Items Overview: Expense Reports Saved Search

For asynchronous Watchlist items, the count is updated only upon request. For example, in Expenses, there is an expense reports search panel from which users can make searches and save them in MDS for future use. Users should be able to promote their saved searches to the Watchlist for tracking. This Watchlist item (of type USER_SAVED_SEARCH) is asynchronous in that the Watchlist count will be updated only upon request, and events that change the count will not be updating the Watchlist simultaneously. In this case, Watchlist code is responsible for querying the count of an asynchronous Watchlist item on demand.

For the Expense report saved search panel example:

- Because this item is of type USER_SAVED_SEARCH, Watchlist items are created when the user promotes a saved search on the search panel in Expenses. This invokes a Watchlist service on the Watchlist that does the Watchlist item creation. This is not needed for Watchlist items of type SEEDED_QUERY or SEEDED_SAVED_SEARCH.

- A request to refresh the Watchlist item count comes from the Watchlist portlet. This will invoke an exposed service, which delegates the call to a method on a provided Watchlist JAR file.
• The method on the Watchlist JAR file will take care of rerunning the query (on the expenses database) and taking the results to update the Watchlist item count (on the Watchlist database).

15.3.3.2 Summary of Implementation Tasks

At a high level, for asynchronous Watchlist items, your development tasks are:

• Determine or set up view objects to execute the query for Watchlist count. You may want to include view criteria with bind variables and specify default values for bind variables. (Not needed for Human Task)

For example, most view objects seeded for the Watchlist would filter by user to show counts specific to the logged-in user. You could create a view criteria with a bind variable called `userId` and specify the default value as a groovy expression to determine the current user ID from the security context. You then would seed this view criteria ID in the setup table along with the view object. The Watchlist API would execute the view object by applying this view criteria to get the row count. Example 15-2 shows code from the view object xml for a bind variable with a default value.

• (Optional) Set up a summary view object to facilitate a refresh count and specify the `summary` attribute in the Watchlist setup table. Watchlist code would use this information to query the count instead of doing a rowcount on the executed view object results.

• Include Watchlist model JAR files in your service project and set up a `refreshWatchlistCategory` service method. This method only will contain code to delegate the call to the nested Watchlist application module method that actually executes the view object queries by reading your Watchlist setup data. This requires that all the corresponding model projects are included as JAR files in this service project so view objects are available in the class path. This service should be exposed so that the Watchlist UI can use it to start the refresh process.

• Determine and code task flows for drilldown from the Watchlist UI. There is no special coding required for these task flows; the key-value parameter string that you specify in the setup table will be used as the input parameter list when invoking the specified task flow for the UI drilldown on Watchlist items.

• (Optional) Enable saved search promotion to the Watchlist. Include Watchlist model JAR files in the corresponding project for the query panel and work with the Watchlist API. For promotion and demotion of user-saved searches, code must invoke a method in the provided Watchlist JAR, which then will handle interaction back to the Watchlist. Add promotion and demotion components on the search panel so that saved searches can be used as Watchlist items.

• Include Watchlist UI and Protected model JAR files in your UI SuperWeb project, to enable Watchlist menu drilldown in the UI Shell Global Area, as shown in Figure 15-10.
Create FND_LOOKUPS for the displayed Watchlist category and item meaning (FND_STANDARD_LOOKUP_TYPES.LOOKUP_TYPE). Product teams must seed the lookup type with the meaning for the category (VIEW_APPLICATION_ID = 0 and SET_ID = 0). The translated lookup type meaning is shown in the Watchlist UI for the category, while the corresponding lookup value meanings are shown in the Watchlist UI for items (user saved search item meanings come from the saved search directly). Product teams must create seed data for this lookup.

Example 15-3 presents sample code to create or update the lookup.

Seed Watchlist categories and setup information. Create your category in ATK_WATCHLIST_CATEGORIES and then seed your items in the reference data table (ATK_WATCHLIST_SETUP). The watchlist_item_type determines how the Watchlist code will handle the item.

Note:

HUMAN_TASK items only require seeding in the tables to work; there are no view objects to create.

Example 15-2  Example Code from the View Object XML for Bind Variable with Default Value

```xml
<Variable
   Name="userId"
   Kind="where"
   Type="java.lang.String">
   <TransientExpression><![CDATA[return adf.context.securityContext.userName;]]></TransientExpression>
</Variable>
```

Example 15-3  Sample Code to Create or Update Watchlist Lookup

```java
declare
begin
   FND_LOOKUP_TYPES_PKG.CREATE_OR_UPDATE_ROW {
      X_VIEW_APPSNAME => 'FND',
      X_LOOKUP_TYPE => 'FIN_EXM_WATCHLIST_CATEGORY',
      X_APPLICATION_SHORT_NAME => 'EXM',
   }
```
15.3.4 How to Use the Watchlist

Follow the procedures described in this section to use the Watchlist.

15.3.4.1 Make the Watchlist Link in the UI Shell Global Area Work

To ensure the Watchlist link works in your pages, complete these steps.

1. Add this ADF Library JAR file to the SuperWeb user interface project for your application (the JAR file must be part of your Web Archive (WAR) in the WEB-INF/lib directory):

   fusionapps/jlib/AdfAtkWatchListPublicUi.jar

2. Add this dependent model ADF Library JAR file in your application (the JAR file must be part of your Enterprise Archive (EAR) in the APP-INF/lib directory):

   fusionapps/jlib/AdfAtkWatchListProtectedModel.jar

3. Add this dependent resource bundle ADF Library JAR file in your application (the JAR file must be part of your EAR in the APP-INF/lib directory):

   fusionapps/jlib/AdfAtkWatchListPublicResource.jar

4. Add these resource-ref entries to the web.xml file in your SuperWeb user interface project:

   ```xml
   <resource-ref>
     <res-ref-name>wm/WorkManager</res-ref-name>
     <res-type>commonj.work.WorkManager</res-type>
     <res-auth>Container</res-auth>
   </resource-ref>
   <resource-ref>
     <res-ref-name>jdbc/ApplicationDBDS</res-ref-name>
     <res-type>javax.sql.DataSource</res-type>
     <res-auth>Container</res-auth>
   </resource-ref>
   ```
15.3.4.2 Seed Reference Data (All items)

See Watchlist Physical Data Model Entities for details of the entire Watchlist data model. Seed only ATK_WATCHLIST_CATEGORIES and ATK_WATCHLIST_SETUP.

15.3.4.3 Create a Summary View Object (SEEDED_QUERY)

- By default, Watchlist code will access the application module or view object that is specified in the setup table, and rerun the query to refresh the Watchlist count. The summary view object is one way to get the count. Usually this will be for efficiency reasons.

- The product team can signal that it wants the Watchlist code to use its summary view object by seeding something in SUMMARY_VIEW_ATTRIBUTE of ATK_WATCHLIST_SETUP. If this is not null, the Watchlist code will get the view object, but instead of running the query, it will take only the first row and get the specified attribute.

- Your task is to create a view object with the correct count in the attribute specified in the setup table.

15.3.4.3.1 Summary Tables

One common reason to use a summary view object is if the seeded query is based on Multiple-Organization Access Control (MOAC) data security. This is because you can calculate the count of this query for each Business Unit ID (BUID). The count of a user is just the sum of the counts of the BUIDs that the user can access.

For example, say you have a table (or query) that has three rows of BUID #1, three rows of BUID #2, and three rows of BUID #3. The current user has access to BUIDs #1 and #2.

If you wanted to get the count, MOAC would filter the rows by BUID and return six rows.

However, because you are interested only in the count, a more efficient way would be to create a summary table for this table. The summary table keeps track of the count for each BUID. For the example table, the summary table would resemble Table 15-4.

<table>
<thead>
<tr>
<th>BUID</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Now, instead of using MOAC to find a count for a particular user, you use MOAC to find the sum of counts for the user. The example user would get the first two rows returned, and you can calculate the total count by summing the count column.

You can use summary tables to populate the summary view object attribute.

15.3.4.4 Create Seeded Saved Searches in MDS (SEEDED_SAVED_SEARCH)

In addition to seeding seeded saved searches in the reference tables, the saved searches need to be seeded in MDS so that when the user visits the saved search panel,
the saved search will show as one of the choices in the drop-down box. Saved searches in MDS are stored in the file system as an XML file for each view object. The steps to create this file are:

1. Ensure you have enabled MDS for your application.
2. Run the application and visit the desired search page.
3. Use the UI to create saved searches and name them. For each saved search, select **Run Automatically**.
4. Examine the `adf-config.xml` file to determine where your file-based MDS repository is located.
5. In a terminal, change to the directory:
   ```
   <MDS repository>/persdef/oracle/apps/.../mdssys/cust/user/
   <user>/
   ```
   The `<user>` directory would be the user you used to save the search with, or anonymous by default.
6. Inside that directory, there should be an `xxxVO.xml.xml` file that contains the created saved searches. Keep that file.
7. In that XML file, note the saved search ID, which should match the View Criteria ID in the reference data.

   **Note:**
   This ID can be different than the display name that you entered in the saved search UI.

15.3.4.5 Creating Application Module and View Objects (All except HUMAN_TASK)
Refer to [Implementing the Watchlist](#).

15.3.4.6 Setting Up Service (All except HUMAN_TASK)
For the same Watchlist items, the Watchlist portlet must be able to invoke a refresh. You will set up and expose a service that includes local JAR files for this purpose. The nested Watchlist JAR file can use those local JAR files in its refresh code.

The service will expose the `refreshCategory` method that will delegate the call to the same method in the nested Watchlist application module. This method will be provided in the Watchlist JAR file and will contain code to perform the category-wide refresh.

15.3.4.7 Importing All Watchlist-Related Application Modules
Your Service project must import the other JAR files from your product that must be used by the Watchlist code.

15.3.4.8 Nesting Watchlist Application Modules
Include `AdfAtkWatchListProtectedModel.jar` and `AdfAtkWatchListPublicModel.jar` in your service project, and nest `AdfAtkWatchListPublicUi.jar` in your service application module.
Set up the AppMasterDB connection that comes with it. Point it to the database where you have seeded your data in the Watchlist tables. This usually is your development database that is used for the ApplicationDB connection.

15.3.4.9 Using the refreshWatchlistCategory Method

This method, shown in Example 15-4, refreshes all Watchlist items in the corresponding category.

**Example 15-4  Refreshing Watchlist Items**

```java
public void refreshWatchlistCategory(String categoryCode) {
    AtkWatchlistPublicAMImpl wlAM = this.getAtkWatchlistPublicAMImpl();
    wlAM.refreshWatchlistCategory(categoryCode);
}
```

15.3.4.10 Importing Watchlist JAR Files into the Saved Search Project (USER_SAVED_SEARCH)

For subsequent steps that require running Watchlist APIs from your code, you must import the Watchlist JAR files. These also contain an AppMasterDB connection that must point to the Watchlist database.

- Add AdfAtkWatchlistProtectedModel.jar and AdfAtkWatchlistPublicService.jar (fusionapps > jlib) as ADF libraries in the appropriate data model projects, preferably in a model project that is visible to both service and user interface model project application modules.
- Add AdfAtkWatchlistPublicUi.jar as an ADF library to your user interface project.
- Configure the AppMasterDB connection.

15.3.4.11 Promote Saved Search to the ATK Watchlist (USER_SAVED_SEARCH)

Every Watchlist-enabled saved search panel must include a component to let the user control which of the saved searches to promote. The pre-seeded saved searches will be shown as static, while the user can use checkboxes to determine which saved searches should be promoted. There will be listeners to this component that will publish business events to make the appropriate worklist changes.

Before You Begin

Ensure that the following steps have been performed.

- Populate ATK tables with SEED watchlist category information. Product teams need to provide information about the service that refreshes the watchlist item count.
- Populate ATK tables with appropriate watchlist setup information. There must be a watchlist setup item of type USER_SAVED_SEARCH.
- Ensure the application is MDS enabled so users can save their searches and that the saved searches exist across sessions.
- From the watchlist UI, users can drill down to the transactional UI flows. Ensure that the transactional UI flow is properly set up so that it can show the search page when the user clicks the Watchlist item in the Watchlist UI.
• In the application project, create a backing bean and register it as a backingBeanScope bean. In the backing bean, create the Java method shown in Example 15-5:

**Example 15-5   Creating the Backing Bean**

```java
import oracle.adf.view.rich.model.QueryDescriptor;
import oracle.adf.view.rich.model.QueryModel;

public List<String> getWatchListUserSavedSearchList() {
    if (AppsLogger.isEnabled(AppsLogger.FINEST)) {
        AppsLogger.write(this, "Watchlist saved search promotion: Entering method getWatchListUserSavedSearchList", AppsLogger.FINEST);
    }
    List<String> WatchListUserSavedSearchList = new ArrayList<String>();
    QueryModel queryModel = (QueryModel) evaluateEL("#{bindings.ImplicitViewCriteriaQuery.queryModel}"); // See next code sample for evaluateEL()
    if (queryModel != null) {
        if (AppsLogger.isEnabled(AppsLogger.FINEST)) {
            AppsLogger.write(this, "Watchlist saved search promotion: queryModel is not null", AppsLogger.FINEST);
        }
        List userQueries = queryModel.getUserQueries();
        if (userQueries != null & userQueries.size() > 0) {
            if (AppsLogger.isEnabled(AppsLogger.FINEST)) {
                AppsLogger.write(this, "Watchlist saved search promotion: User Saved Searches exist", AppsLogger.FINEST);
            }
            for (int i = 0; i < userQueries.size(); i++) {
                QueryDescriptor qd = (QueryDescriptor) userQueries.get(i);
                if (qd != null) {
                    if (AppsLogger.isEnabled(AppsLogger.FINEST)) {
                        AppsLogger.write(this, "Watchlist saved search promotion: Adding user saved search name to the watchListUserSavedSearchList using QueryDescriptor getName: " + qd.getName(), AppsLogger.FINEST);
                    }
                    WatchListUserSavedSearchList.add(qd.getName());
                }
            }
        }
    }
    if (AppsLogger.isEnabled(AppsLogger.FINEST)) {
        AppsLogger.write(this, "Watchlist saved search promotion: Exiting method getWatchListUserSavedSearchList: returning watchListUserSavedSearchList: " + WatchListUserSavedSearchList, AppsLogger.FINEST);
    }
    return WatchListUserSavedSearchList;
}
```

**15.3.4.11.1 How to Promote a User-Saved Search to the Watchlist**

Follow these steps to integrate the ATK task flow to promote saved searches to the Watchlist.
1. Ensure the AdfAtkWatchListPublicUi.jar file is available (usually in the fusionapps/jlib directory).

2. Start JDeveloper and open the .jspx or .jsff page containing the query region whose saved searches have to be promoted.

3. In the query region, add a toolbar facet.
   a. Right-click the component.
   b. In the menu that opens, select Facets-Query.
   c. From the submenu, select Toolbar, as shown in Figure 15-11.

![Figure 15-11    Adding Toolbar Facet to Query Region](image)

3. In the toolbar facet of the query region, drag and drop an ADF Toolbar component, such as Toolbar (ADF Faces.Common Components) shown in Figure 15-12, onto the page.

![Figure 15-12    Adding an ADF Toolbar Component](image)

The toolbar facet in the Source view will look similar to:

```xml
<f:facet name="toolbar">
  <af:group id="g1">
    <af:toolbox id="t1">
      <af:region value="#{bindings.AtkWatchlistUserSavedSearchPromotionTF1.regionModel}" id="r7"/>
    </af:toolbox>
  </af:group>
</f:facet>
```

4. Open the Resource Palette and create a File System connection to the directory containing the AdfAtkWatchListPublicUI.jar file, as shown in Figure 15-13.
6. Expand the connection node and the ADF library node in the Resource Palette as shown in Figure 15-14.

Figure 15-14  Expanding the Connection Node

When the ADF Task Flows node is expanded, you should see two task flows. The task flow AtkWatchlistUserSavedSearchPromotionTF is the one to be used.

7. Drag and drop the AtkWatchlistUserSavedSearchPromotionTF task flow as a region into the toolbar component (present in the query region toolbar facet), created in the previous steps. As soon as the task flow is dropped onto the page, the Edit Task Flow Binding dialog is displayed. Enter the following values for the mandatory parameters.

- **categoryCode**: Provide the WATCHLISTCATEGORY_CODE that has been seeded in the ATK tables.
- **watchlistItemCode**: Provide the WATCHLISTITEM_CODE provided while creating the Watchlist setup data.
- **userSavedSearchList**: This represents the model object of the query region. To populate this field:
  a. Select the userSavedSearchList input field, click the small “v” icon present at the end of the field and select the Expression Builder option.
b. Select the value from the Java method shown in Example 15-5. In this case, the value is `watchListUserSavedSearchList`, found in `ADF Managed Beans > backingBeanScope > searchOrderScheduleBackingBean > watchListUserSavedSearchList`.

The Expression will be:

```java
#{backingBeanScope.searchOrderScheduleBackingBean.watchListUserSavedSearchList}
```

c. Click OK to insert the value in the `userSavedSearchList` field.

- **internalCriteriaName**: This represents the ViewCriteria Name of the search binding executable of the query region present in the UI page. To populate this field, follow the same steps as you did to populate the `userSavedSearchList` field, but select `internalCriteriaName`. Also see Additional Steps for Existing Consumers.

When you are finished, the Edit Task Flow Binding dialog will resemble Figure 15-15.

**Figure 15-15 Completed Edit Task Flow Binding Dialog**

![Edit Task Flow Binding dialog](image)

8. Click OK. This creates a region component in the UI page, as shown in Figure 15-16.

**Figure 15-16 Created Region Component**

![Created Region Component](image)

9. Open the page definition file and select the executable associated with the Watchlist-related task flow.

10. Open the Property Inspector and set the Refresh field to `ifNeeded`, as shown in Figure 15-17. The ATK saved search promotion task flow has to be refreshed each time the query model changes. This step ensures that the task flow is refreshed whenever the query model changes.
11. Add a task flow security permission to the AtkWatchlistUserSavedSearchPromotionTF task flow in the jazn-data.xml file.

   a. Open the jazn-data.xml file and select the ADF Policies tab.

   b. From the Task Flow list, select AtkWatchlistUserSavedSearchPromotionTF, grant it to an appropriate role, and select appropriate actions as shown in Figure 15-18.

12. Run the UI page. In the toolbar facet of the query region, there will be a Watchlist Options button, as shown in Figure 15-19.

   When you click the button, a popup with the list of all saved searches is displayed, as shown in Figure 15-20.
Additional Steps for Existing Consumers

If you already have implemented promoting Saved Search to the ATK Watchlist, there are four additional steps.

1. In your pageDef, change the WatchList task flow parameter called queryModel from:
   
   `<parameter id="queryModel" value="#{bindings.ExistingCriteria.queryModel}"/>
   
   to:
   
   `<parameter id="userSavedSearchList" value="#{backingBeanScope.YourBean.watchListUserSavedSearchList}"/>

2. Use the public List<String> getWatchListUserSavedSearchList() Java method by passing the QueryModel binding of your af:query.

3. Change the old queryBinding parameter to the new internalCriteriaName parameter. For example, change:
   
   `<parameter id="queryBinding" value="#{bindings.SearchPageVOCriteriaQuery}"/>
   
   to:
   
   `<parameter id="internalCriteriaName" value="#{backingBeanScope.searchRelatedBean.internalCriteriaName}"/>

4. In someBackingBeanScopeBean.java, such as the one you created in Example 15-5, add the two methods shown in Example 15-6.

   Note: In someBackingBeanScopeBean.getInternalCriteriaName(), the queryBinding variable in the first line is the one you see in af:query. For example, af:query
   
   queryListener="#{bindings.SearchPageVOCriteriaQuery.processQuery}"
   
   Just take the QueryBinding,
   
   #{bindings.SearchPageVOCriteriaQuery}.

**Example 15-6  Additions to the backingBeanScopeBean**

```java
import javax.el.ExpressionFactory;
import javax.el.MethodExpression;
import javax.el.ValueExpression;
import javax.faces.context.FacesContext;
import oracle.adf.model.binding.DCBindingContainer;
import oracle.jbo.uicli.binding.JUSearchBindingCustomizer;

public Object evaluateEL(String expr)
{
    FacesContext facesContext=FacesContext.getCurrentInstance();
    ExpressionFactory exprFactory=facesContext.getApplication().getExpressionFactory();
    ValueExpression valueExpr=exprFactory.createValueExpression(facesContext.getELContext(), expr, Object.class);
    return valueExpr.getValue(facesContext.getELContext());
}

public String getInternalCriteriaName() {
    if (AppsLogger.isEnabled(AppsLogger.FINEST)) {
        AppsLogger.write(this, "Watchlist saved search promotion: Entering method getInternalCriteriaName", AppsLogger.FINEST);
    }
    return 
```
String queryBinding = "#{bindings.SearchPageVOCriteriaQuery}"
DCBindingContainer searchBinding = (DCBindingContainer)evaluateEL(queryBinding);
internalCriteriaName = JUSearchBindingCustomizer.getCriteriaName(searchBinding);
if (AppsLogger.isEnabled(AppsLogger.FINEST)) {
    AppsLogger.write(this, "Watchlist saved search promotion: Exiting method
getInternalCriteriaName with return value for internalCriteriaName: " +
    internalCriteriaName, AppsLogger.FINEST);
}
return internalCriteriaName;
}

15.3.4.12 Code Task Flows to Accept Parameters (All except HUMAN_TASK)

If you have seeded the information properly, your normal task flows should work
when expanded into from the Watchlist portlet.

15.3.4.12.1 Saved Search

For saved search Watchlist items, you will want the drilldown to load a specific saved
search by default.

One function of the Watchlist portlet will be to take a user to the corresponding action
area when the user clicks a Watchlist item. For saved searches, the desired
functionality is to open the task flow containing the saved search panel and by default,
show the clicked saved search.

On the portlet, upon clicking the link, code will put the proper ViewCriteria name
into the PageFlowScope with the parameter name vcName.

When loaded, the destination task flow will have two tasks:

- Look into the PageFlowScope to retrieve the ViewCriteriaName.
- Obtain the RichQuery object, and apply the ViewCriteriaName to it, if one is
given.

You will be concerned with implementing the two steps for the destination task flow.
First, you can retrieve the ViewCriteriaName from the PageFlowScope with the
code shown in Example 15-7.

Second, you can use the code shown in Example 15-8 to apply the ViewCriteria to
the search panel. If no ViewCriteria was passed, the code loads the default
ViewCriteria.

This code should be run once upon loading the destination task flow. One solution is
to connect it to the rendered property of the search panel, and use a static variable to
ensure that it only runs once.

Example 15-7 Retrieving the ViewCriteriaName from the PageFlowScope

Map pfs = RequestContext.getCurrentInstance().getPageFlowScope(); String vcName =
(String) pfs.get("vcName");

Example 15-8 Applying ViewCriteria to the Search Panel

if (vcName == null || vcName.equals("")) {
    // If no ViewCriteria is given, load default VC
    DCBindingContainer dcbbc = (DCBindingContainer)BindingContext.getCurrent()
        .getCurrentBindingsEntry();
    FacesCtrlSearchBinding fcsb =
        (FacesCtrlSearchBinding)dcbbc
Implement Search Functions in the UI Shell 15-41
.findExecutableBinding("ImplicitViewCriteriaQuery");
FacesCtrlSearchDef def = (FacesCtrlSearchDef)fcsb.getDef();
DCParameterDef paramDef =
(DCParameterDef)def.getParameterDef(
JUSearchBindingCustomizer.PARAMCRITERIA);
fcsb.evaluateParameter(paramDef.getExpression(),false));
} else {
    QueryModel model = search_query.getModel();
    QueryDescriptor selDescriptor = model.create(vcName, null);
    if (selDescriptor != null) {
        model.setCurrentDescriptor(selDescriptor);
    }
    BindingContext.bindings =
    BindingContext.getCurrent().getCurrentBindingsEntry();
    OperationBinding method = (OperationBinding)
    bindings.getControlBinding("applyViewCriteriaByName");
    method.getParamsMap().put("name", vcName);
    method.execute();
}

15.3.4.13 Import Watchlist UI JAR File in User Interface Project

There is a link in the UI Shell for Watchlist in the global area. To make it work, the
user interface project must include these Watchlist UI JAR files:
AdfAtkWatchListPublicUI and its dependent model JAR file
AdfAtkWatchListProtectedModel.

15.3.4.14 Additional Entries for Standalone Deployment

These entries are required for the Watchlist service and UI to be able to work in a
standalone deployment.

- Add this entry in the ejb-jar.xml file in your service project that contains the
  watchlist service next to the similar entry for ApplicationDB. You need
  resource-ref entries for both ApplicationDB and AppMasterDB:

```xml
<resource-ref>
    <res-ref-name>jdbc/AppMasterDBDS</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

- Add this entry in the web.xml file in your SuperWeb project next to the similar
  entry for ApplicationDB. You need resource-ref entries for both
  ApplicationDB and AppMasterDB:

```xml
<resource-ref>
    <res-ref-name>jdbc/AppMasterDBDS</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
</resource-ref>
```

- The connections.xml file should have a valid database entry for
  AppMasterDB.

15.4 Implementing Group Spaces

Group Spaces bundle all the collaboration tools and provide an easy way for users to
create their own ad hoc collaborative groups around a project or business artifact.
This section describes how to implement the Group Spaces functionality that is available in UI Shell.

Assumptions
These assumptions are made:

- The implementation is occurring in a label that is either dependent on ATGPF_MAIN or uptakes ATGPF_MAIN on a regular basis.
- The spaces application and the JDeveloper Standalone WebLogic Server that would run the application have the requisite setup done.
- The consuming applications are secure. Group Spaces functionality attempts to retrieve the group spaces for the logged-on user. Without a secure application, this functionality would fail.

15.4.1 How to Implement Group Spaces

Follow these steps to implement Group Spaces.

1. Ensure the Oracle WebCenter Portal Spaces Client library, spaces-webservice-client.jar, has been added to the Project.

2. Define an application connection to point to the URL of the WebCenter Portal Spaces WebService. To do this:
   a. Right-click Connections in the Application Resource palette.
   b. Select New Connection > URL.
   c. Enter the value $HOST:$PORT/webcenter/SpacesWebService, where $HOST and $PORT are the hostname and the port on which the spaces application is running.
   d. Save this connection with the name SpacesWebServiceEndpoint.

3. To make a homepage tab appear within UI Shell, deploy the Functional Setup Manager application.

15.4.2 Overview of Group Spaces Functionality

The Group Spaces functionality implements these features:

- When the Group Spaces link is clicked, a popup displays the logged-in user's Group Spaces.
- When the user clicks a Group Space from the list, the Group Space's home page is opened in an iFrame. This iFrame is rendered within a Home Page tab called WebCenter. The Group Space is opened suppressing the WebCenter Portal chrome but will still render all the tabs within that Group Space. (Note: Chrome is a term for the visible graphical interface features of an application.) This chrome level suppresses the WebCenter Portal chrome but will still render all the tabs within that Group Space.
- When the user clicks View All Group Spaces, the same UI as the My Group Spaces in the spaces application is rendered. This is also rendered as an iFrame within the WebCenter Portal HomePage tab where it suppresses the chrome as well as the top level WebCenter Portal tabs.
15.4.3 How to Pass a Chromeless Template

When navigating to the WebCenter Portal home page, a WebCenter Portal template that does not contain the header chrome is desired. This can be done by appending &wc.pageTemplate=oracle.webcenter.spaces.siteTemplate.gsContent.

Pass this parameter when navigating to a Group Space from the Group Spaces global dialog, Tag Center, Global Search, or an Activity Stream link.

Use a question mark (?) instead of an ampersand (&) if this is the only request parameter for that URL.

15.5 Implementing Activity Streams and Business Events

Activity Streams is a feature provided by the WebCenter Portal. Use Activity Streams to capture changes and publish Activity messages. Customer Relations Management (CRM), in particular, makes heavy use of this feature to keep abreast of service requests and opportunities. Users subscribe to Activity Streams by using the Activity Streams user interface.

For business events, Activities are shown only to users who subscribe to the stream and who have the necessary security access.

Activity Streams can be connected to:

- **Business Objects.** At a high level, business objects correspond to a workarea, such as Sales or Contacts.

- **Group Spaces.** For example, a team lead can set up a Group Space that team members can use to share information about a project.

- **People Connection.** This is similar to the various social networking sites on the Internet that let people interact with friends and business associates.

This section is concerned only with business objects.

15.5.1 Introduction to WebCenter Portal Activities

A WebCenter Portal Activity is comprised of the following:

- **Actor** - The user who performed the action that triggered the business event. For Oracle Fusion Applications, this will be the userid fetched from the user session.

- **ActivityType** - The type of Activity to be published. This defines the format of the Activity message.

- **Objects** - The objects associated with the Activity. There could be multiple objects associated with an Activity, but for business events, only the event source is used as an object.

WebCenter Portal Activities are defined in the service-definition.xml file. The scope of service_definition.xml is per business object. The service ID attribute should match the name of the entity object. The service-definition.xml file contains ActivityTypes, ObjectTypes and resource-view definitions. An ActivityType must be defined for every business event on the entity object. The type name should match the business event's name. The messageFormatKey attribute in the ActivityType element points to a message key in a Resource Bundle.
It defines the format of the message displayed in the Activity Stream UI. These tokens are supported in a message.

- `{actor[0]}`: Replaced by the display name of the user who triggered the event.
- `{object[0]}`: Replaced by the value of the attribute in the event payload whose attribute name matches the object type name.
- `{custom[attr1].value}`: Replaced by the value of the `attr1` attribute in the event's payload.

The message format would look similar to:

```
{actor[0]} updated Opportunity {object[0]} status to {custom[status].value}
```

### 15.5.2 How to Publish Business Events to Activities

ADF Business Components business events are, by default, published to SOA Event Delivery Network (EDN). Applications Core implements a `BusinessEventAdapter` to listen to these events, transition them to Activities, and asynchronously publish them to the ActivityStream Service. This adapter is a singleton per application and publishes the business events raised to the `ActivityService` in the order they are produced. A business event is published as an Activity only when an `ActivityType` matching the name of the event is found in the service definition for the business object.

While mapping a business event to an Activity, keep these notes in mind:

- There is one-to-one mapping between an Activity Service definition and a business object. The service ID attribute in the `service-definition.xml` file should match the Entity name.
- The `ActivityType` name should match the name of the business event.
- Define an `ObjectType` with the name attribute matching an attribute name in the payload. This attribute value will replace the `{object[0]}` token in the message format. WebCenter Portal supports multiple object types for an Activity, but for business events-related Activities, only one Object that corresponds to the event source is supported. The Object type name should match the name of the attribute whose value should be displayed in the hyperlink for the object.
- Define a message format using tokens for `Actor`, `Object` and `customAttributes`. The custom attribute names used in the token should match the attribute names in the payload.
- In the Activity message displayed in the UI, only `Actor` and `Object` display values will be rendered as hyper links. If an Activity involves multiple objects, hyper links will be supported only for the event source object. Multiple attributes from the event payload can be referenced in the message.
- The hyperlink for the object allows users to navigate to the business object’s work area. The target page for navigation can be configured through the `resource-view` element in the `service-definition.xml` file.

### 15.5.3 How to Publish Activities Using a Programmatic API

For certain scenarios, Oracle Fusion Applications are required to publish Activities for model changes that are not based on entity objects. Since Oracle ADF Business Components business events are based on entity objects, it is not possible to use these
events for publishing Activities for non entity object-based model changes. For such scenarios, product teams could use the Applications Core API shown in Example 15-9 to programmatically publish Activities to the ActivityStream service.

**BusinessActivityPublisher**

This class provides the publishActivity API that can be used to publish Activities asynchronously. This is a singleton per Java EE application. An instance of this class can be obtained using the getInstance API. This lets product teams define such things as ActivityTypes, ObjectTypes, and resource-view definitions, declaratively in the service-definition.xml file, similar to business event-related activities. This would allow product teams to follow the same mechanism to define and publish Activities for both entity object and non-entity object-based model changes with very little code changes.

**BusinessActivity Class**

This is an abstract class that is used to represent an Activity corresponding to a business event. BusinessActivityPublisher:publishActivity() takes an instance of this class as a parameter. You would implement an instance of this class to encapsulate the details of the Activity corresponding to the non entity object-based model changes and invoke the publishActivity API with this as a parameter. BusinessActivityPublisher will find the matching ActivityType and ObjectType defined for this Activity in service-definition.xml and publish the Activity to the ActivityStreaming Service asynchronously. Details of the API on this class are shown in Example 15-10.

**BusinessActivity**

This is an abstract class that is used to represent an Activity corresponding to a business event. BusinessActivityPublisher:publishActivity() takes an instance of this class as a parameter. You would implement an instance of this class to encapsulate the details of the Activity corresponding to the non entity object-based model changes and invoke the publishActivity API with this as a parameter. BusinessActivityPublisher will find the matching ActivityType and ObjectType defined for this Activity in the service-definition.xml file and publish the Activity to the ActivityStreaming Service asynchronously. Details of the API on this class are shown in Example 15-11.

**Example 15-9  BusinessActivityPublisher.java**

```java
/**
 * This class is responsible for publishing business events as WebCenter Portal
 * Activities to the ActivityStreaming service. This is a singleton per Java EE
 * application. An instance of this object is obtained using the getInstance() 
 * method. This class transforms business events into Activities and publishes them
 * to Activity Service asynchronously. Resources held by the class are released by
 * using release() method. In Java EE container, release is done by
 * Applications Core when the application is undeployed or stopped.
 */

public class BusinessActivityPublisher
{
  /**
   * Returns and instance of BusinessActivityPublisher if one exists or
   * creates a new one.
   *
   */

  public synchronized static BusinessActivityPublisher getInstance()

  /**
   * Queues the Activity for publishing. The queued activities are published
   */
```
* to the Activity Streaming service asynchronously if there is a matching
* ActivityType defined for the source business event. If no matching
* ActivityType is found in the service-definition.xml corresponding to the
* source entity object, the activity is ignored.
* @param activity
 */
public void publishActivity(BusinessActivity activity)

/**
 * Should be called during App undeploy to stop the publisher thread.
 * In a Java EE container, this method is called by Applications Core
 * ServletContextListener.
 */
public void release()

Example 15-10  BusinessActivity.java

/**
 * Name of the ActivityType defined in service-definition that
 * corresponds to serviceId returned by getServiceId().
 * @return Name of the ActivityType
 */
public String getName()

/**
 * ID of the service-definition containing the metadata for this
 * Activity.
 * @return serviceId
 */
public abstract String getServiceId();

/**
 * Array of GUIDs for Actors of the Activity.
 * @return array of guids for the Actors.
 */
public abstract String[] getActors();

/**
 * This api will return additional service ids
 * @return Array of ServiceIds
 */
public String[] getAdditionalServiceIds(){ return null;}

/**
 * This attr provides a "," separated list of object type
 * names associated with a particular Activity.
 * @return
 */
protected String getActivityObjectTypeNames(){
    return null;
}

/**
 * Payload for the Activity. Every attribute that is part of this payload is
 * persisted in WC as custom attribute of the Activity Object so only
 * attributes needed for the Activity Message should be added to the payload
 * to avoid performance overhead.
 * The payload typically contains:
 * 1. Attribute(s) whose name matches the object-type name attribute in
 *    service-definition. This value is used in generating the object-id
 *    of the object referenced in the Activity Stream message.
2. All the attributes referenced in the Message format using {custom} token.
   @return a map containing the attribute names and their values needed to
display the Activity Message for this Activity.
*/
public abstract Map getPayload();

Example 15-11 BusinessActivity.java

/**
 * Name of the ActivityType defined in service-definition that
 * corresponds to serviceId returned by getServiceId().
 * @return Name of the ActivityType
 */
public String getName()

/**
 * ID of the service-definition containing the metadata for this
 * Activity.
 * @return serviceId
 */
public abstract String getServiceId();

/**
 * Array of GUIDs for Actors of the Activity.
 * @return array of guids for the Actors.
 */
public abstract String[] getActors();

/**
 * This api will return additional service ids
 * @return Array of ServiceIds
 */
public String[] getAdditionalServiceIds(){ return null;}

/**
 * This attr provides a "," separated list of object type
 * names associated with a particular Activity.
 * @return
 */
protected String getActivityObjectTypeNames(){
  return null;
}

/**
 * Payload for the Activity. Every attribute that is part of this payload is
 * persisted in WC as custom attribute of the Activity Object so only
 * attributes needed for the Activity Message should be added to the payload
 * to avoid performance overhead.
 * The payload typically contains:
 * 1. Attribute(s) whose name matches the object-type name attribute in
   service-definition. This value is used in generating the object-id
   of the object referenced in the Activity Stream message.
 * 2. All the attributes referenced in the Message format using {custom}
   token.
 * @return a map containing the attribute names and their values needed to
display the Activity Message for this Activity.
 */
public abstract Map getPayload();
15.5.4 How to Implement Activity Streams

This section provides details about the steps involved in integrating this feature with Oracle Fusion Applications.

15.5.4.1 Defining and Publishing Business Events in JDeveloper

To define a business event, follow these steps:

1. In the Application Navigator, double-click an entity object.
2. In the overview editor, click the business events navigation tab.
3. On the business events page, expand the Event Publication section and click the Edit event publications icon.
4. In the Edit Event Publications dialog, click New to create a new event.
5. Double-click the new cell in the Event column, and select the appropriate event.
6. Double-click the corresponding cell in the Event Point column, and select the appropriate event point action.
7. You optionally can define conditions for raising the event using the Raise Conditions table.
8. Click OK.

An event definition in the Entity XML file would look similar to:

```
<EventDef Name="OpportunityStatusUpdate">
  <Payload>
    <PayloadItem AttrName="OpptyId"/>
    <PayloadItem AttrName="Status"/>
    <PayloadItem AttrName="Customer.CustomerId"/>
  </Payload>
</EventDef>
```

15.5.4.2 Overriding isActivityPublishingEnabled() to Enable Activity Publishing

By default, business events are not published as Activities. You should override the isActivityPublishingEnabled() method to enable Activity publishing for an entity object. Table 15-5 shows the details about the APIs exposed in OAEntityImpl that you can override.

Note that, except for the isActivityPublishingEnabled() method, other methods mentioned in Table 15-5 should be avoided in favor of transient attributes specified in Defining Activity Attributes Declaratively.
<table>
<thead>
<tr>
<th>Method</th>
<th>Return Type</th>
<th>Description</th>
<th>Optional/ Required</th>
<th>Corresponding Declarative Transient Attribute (See Defining Activity Attributes Declaratively)</th>
</tr>
</thead>
<tbody>
<tr>
<td>isActivityPublishingEnabled()</td>
<td>boolean</td>
<td>By default the base class implementation returns false. This will enable activity publishing for this entity object.</td>
<td>Required</td>
<td>None</td>
</tr>
<tr>
<td>getActivityActorsGUIDs()</td>
<td>String []</td>
<td>This can be overridden to provide an array of GUIDs for the Actors involved with the activity. By default, the framework will use the GUID of the user currently logged in when the business event is raised.</td>
<td>Optional</td>
<td>WCActivityActorGuid1, WCActivityActorGuid2</td>
</tr>
<tr>
<td>getActivityStreamServiceId()</td>
<td>String</td>
<td>This returns the service ID to be used to publish the activities for the business events raised for this entity. By default this returns null. When null is returned, the service ID defaults to the full name of the entity object.</td>
<td>Optional</td>
<td>WCActivityStreamServiceId</td>
</tr>
<tr>
<td>getAdditionalServiceIds()</td>
<td>String []</td>
<td>Override this API to support publishing multiple Activities in response to a single event. So additional service IDs can be passed for a single activity.</td>
<td>Optional</td>
<td>WCAdditionalActivityServiceId1, WCAdditionalActivityServiceId2</td>
</tr>
<tr>
<td>getActivityObjectTypeNames()</td>
<td>String</td>
<td>This API can be overridden to return multiple object type names. Values provided should be a comma-separated list of object type names associated with a particular Activity. The first object-type in this string will be used to create the custom attributes needed for the primary object. When creating the primary object for an Activity, the object-type of the first object listed in the service-definition.xml file is necessary even though the custom attributes used to construct this object are fetched from a different object-type based on get(ActivityObjectTypeNames) or WCActivityObjectTypeNames(). This is necessary for the follow model to work. The order of the objects in this array can be used to reference the objects in an Activity message format string.</td>
<td>Optional</td>
<td>WCActivityObjectTypeNames</td>
</tr>
</tbody>
</table>
15.5.4.3 Defining Activity Attributes Declaratively

Some of the attributes, such as Actor, Service Ids and Additional Service Ids, can be passed as a part of the payload. The basic process steps are:

- Define a transient attribute in the entity object.
- Give a default value to the transient attribute.
- Include the transient attribute as a part of the payload.

The different transient attributes that can be passed with the payload are shown in Table 15-6.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCActivityServiceId</td>
<td>String</td>
<td>This attribute’s value is used to identify the business object service to be associated with the Activity.</td>
<td>oracle.apps.crmdeemo.model.OpportunityEO</td>
</tr>
<tr>
<td>WCAdditionalActivityServiceId1</td>
<td>String</td>
<td>This attribute’s value is used to identify any additional service that must be associated with the Activity.</td>
<td>Values are similar to those of WCActivityServiceId</td>
</tr>
<tr>
<td>WCAdditionalActivityServiceId2</td>
<td>String</td>
<td>This attribute’s value is used to identify any additional service that must be associated with the Activity.</td>
<td>Values are similar to those of WCActivityServiceId</td>
</tr>
<tr>
<td>WCActivityActorGuid1</td>
<td>String</td>
<td>This attribute’s value is used to identify any actor that must be associated with the Activity if the default actor value must be overridden. For instance, in the message it can be accessed as actor[0]</td>
<td>&lt;User_GUID&gt;</td>
</tr>
<tr>
<td>WCActivityActorGuid2</td>
<td>String</td>
<td>This attribute’s value is used to identify any actor that must be associated with the Activity if the default actor value must be overridden. For instance, in the message it can be accessed as actor[1]</td>
<td>&lt;User_GUID&gt;</td>
</tr>
</tbody>
</table>
Table 15-6  (Cont.) Transient Attributes that Can Be Passed with the Payload

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
<th>Sample Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCActivityObjectTypeNames</td>
<td>String</td>
<td>This attribute should provide a comma-separated list of object-type names associated with a particular Activity. Programmatically getActivityObjectTypeNames() API in the entity object’s EntityImpl. The first object-type in this string will be used to create the custom attributes needed for the primary object. When creating the primary object for an Activity, the object type of the first object listed in the service-definition.xml file is necessary even though the custom attributes used to construct this object are fetched from a different object type based on getActivityObjectTypeNames() or WCActivityObjectTypeNames. This is necessary for the follow model to work. The order of the objects in this array can be used to reference the objects in the Activity message format string.</td>
<td>Emp, Dept</td>
</tr>
</tbody>
</table>

15.5.5 How to Define Activities

Defining Activities requires:

- Adding the ActivityStream UI task flow
- Defining Activities in the service-definition.xml file

15.5.5.1 Add the ActivityStream UI Task Flow

To add the ActivityStream task flow:

1. Ensure your user interface project includes the Oracle WebCenter Portal Activity Streaming Service library in the Libraries and Classpath section in the project properties dialog.

2. From Resource Catalog > TaskFlows, drag and drop either an "Activity Stream" or "Activity Stream Summary - View" task flow onto the page where you want to display the Activity Stream UI.

3. Set the taskFlow resourceId parameter to #{securityContext.userName}. This will tie the task flow to the current user at runtime.
4. For additional details about the Activity Stream task flow, see the chapters in the Working with the People Connections Service partition in the Developing WebCenter Portal Assets and Custom Components with Oracle JDeveloper.

15.5.5.2 Define Activities in the service-definition.xml File

The default location of the service-definition.xml file is under META-INF in the project. For Oracle Fusion Applications, this file will be stored in MDS so you need to put it into a directory that can be added to your Metadata Archive (MAR) file.

The standardized location that all applications should use is:

```xml
<app>/<lba>/<product>/*project*//*meta/oracle/apps/meta*/<lba>/<product>/service-definition.xml
```

The name must be unique, such as:

```xml
```

To define Activities in the service-definition.xml file, follow these steps.

1. If necessary, add the directory to the application’s MAR profile. To add a MAR, select Application > Application Properties > Deployment. In the dialog that is displayed, select the MAR file and click Edit.

2. In the Edit dialog, select User Metadata and click Add. Browse to the metadata directory that was just added and click OK.

3. Set the id attribute on the service-definition element to the entity object name for which you want to define the Activities.

```xml
<service-definition xmlns="http://xmlns.oracle.com/webcenter/framework/service" id="oracle.apps.crmdemo.model.OpportunityEO" version="11.1.1.0.0">
```

4. Define an activity type for every business event in the entity object for which you want to display the Activities in the Activity Stream UI. Ensure the event name of the entity object matches the activity-type name attribute value.

```xml
<activity-types>
  <generic-activity-category name="UPDATE">
    <activity-type name="OpportunityStatusUpdate" displayName="Opportunity Status Update" description="Opportunity Status Update" messageFormatKey="OPPTY_STATUS_UPDATED" iconURL="" defaultPermissionLevel="SHARED"/>
  </generic-activity-category>
</activity-types>
```

5. Set message format strings.

Each activity-type should have a message format defined. The message format string can be translated and is stored in a Resource Bundle or an XLIFF bundle. Oracle Fusion Applications uses XLIFF bundles to store the Activity message format strings. The activity-type element in the service-definition.xml file has messageFormatKey attributes that are used to refer to the format strings in the XLIFF bundle.

Activity Stream supports only Java Resource Bundles. The Common String Repository is used for the message format strings.
These attributes are supported on the activity-type element:

- `messageFormatKey` - Used on the Activity Stream full view task flow.
- `summaryByListMessageFormatKey` - Used in the summary view Activity Stream task flow.
- `summaryByCountMessageFormatKey` - Used in the summary view task flow.

**messageFormatKey**

The value of this attribute points to the key defined in the Resource Bundle. These tokens are supported in the message format string.

- `{actor[0]}`: Replaced by the display name of the user who triggered the event.
- `{object[0]}`: Replaced by the value of the attribute in the event payload whose attribute name matches the object type name.
- `{custom[attr1].value}`: Replaced by the value of the `attr1` attribute in the event's payload.

The following sample uses the Java Resource Bundle class:

```
<activity-types>
  <generic-activity-category name="OPPTYUPDATE">
    <activity-type name="OpptyStatusUpdate"
      displayName="OPPTY_UPDATE"
      description="OPPTY_UPDATE_DESCRIPTION"
      messageFormatKey="OPPTY_STATUS_UPDATED"
      defaultPermissionLevel="SHARED"/>
  </generic-activity-category>
</activity-types>
```

In OpportunityResourceBundle, the `OPPTY_STATUS_UPDATED` key is defined as:

```
{"OPPTY_STATUS_UPDATED", "{actor\[0\]} updated {object\[0\]} status to {custom\['status'\].value}"}
```

**summaryByListMessageFormatKey** and **summaryByCountMessageFormatKey**

These attributes are used only when the Activity Stream summarized view task flow is used. The summarized view task flow is used on the portrait page in My Activities and Network Activities mini cards. In summarized view, the Activity messages are summarized or grouped based on an Activity Type. For instance, if multiple Activities of the same type are published, they are combined and displayed as a single Activity message. Within the group of Activities of the same type, the following algorithm is used to generate summarized messages:

a. Summarize activities by finding a common object referenced in the Activity.

b. Summarize or aggregate the Actors either by listing them if there are three or fewer, or by counting them if there more than three.
c. For remaining activities, summarize by finding a common Actor. Summarize or aggregate the objects either by listing them if there are three or fewer, or by counting them if there are more than three.

For example, note the following activities:

b. Viju updated Project Alpha tasks.
c. Ling updated Project Alpha tasks.
d. Monty updated Oppty 200 laptops status
e. Monty updated Oppty Solaris workstations status
f. Monty updated Oppty 2 DB machines status.

In the summarized view, the Activities are summarized as follows:

• Activities a-c: James, Viju, Ling updated Project Alpha tasks.
• Activities d-f: Monty updated 200 laptops, Solaris workstations, 2 DB machines opportunities status.

Example 15-12 shows sample format strings for the preceding scenario.

In the OpportunityResourceBundle, the keys are defined as:

"OPPTY_STATUS_UPDATED_SUMMARY_LIST", "{actor\[0\]} updated status for opportunity {object\[0\]}"
"OPPTY_STATUS_UPDATED_SUMMARY_CNT", "{actor\[0\]} updated {object\[0\].count} opportunities status"

6. Define an object type for the entity object that is the source of the events. Even though Oracle WebCenter Portal supports multiple object types, for Oracle Fusion Applications, only one object type that corresponds to the source of the events is supported. The value of the name attribute should match the name of the attribute in the business event’s payload. This attribute’s value will be used as the display name of the object when displayed in the Activity message. The primary key of the event source will be used as the object ID.

7. The ObjectType custom attributes can be used to provide additional metadata for handling business object references in Activity Stream messages. The custom attributes shown in Table 15-7 will be used.
### Table 15-7  ObjectType Custom Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service-ref-id</td>
<td>ID of the service-definition of a business object referenced in the Activity message of the current business object. This is used when an Activity message contains multiple object references and is used to reference the serviceId of some other business object.</td>
</tr>
<tr>
<td>object-id-attr</td>
<td>The name of the attribute in the business event payload that is necessary as the object-id for an Activity object. Typically this corresponds to the primary key attribute of a business object. If this attribute is not specified, the object-type element's name attribute is used as the default.</td>
</tr>
<tr>
<td>display-name-attr</td>
<td>The name of the attribute in the business event payload that is necessary as the display name of the Activity object. This attribute's value will be used to replace the {object} token in the Activity's message format.</td>
</tr>
</tbody>
</table>

```xml
<object-type>
  <custom-attributes>
    <custom-attribute name="service-ref-id" defaultValue="oracle.apps.crm.model.OpptyEO"/>
    <custom-attribute name="object-id-attr" defaultValue="opptyId"/>
    <custom-attribute name="display-name-attr" defaultValue="opptyName"/>
  </custom-attributes>
</object-type>
```

8. Define resource view handler parameters to allow custom navigation for links rendered in the Activity message. Navigation from the Actor link in the Activity message navigates to the user's portrait page. Custom navigation to the business object workarea is supported. Important fields include:

- `taskFlowId`: The task flow where you want to go.
- `resourceParamList`: The list of parameters that you want to pass to the task flow. For example, if a business object task flow takes the `opptyId` parameter, in `resourceParamList` you should specify "opptyId". If multiple parameters are required, the parameters should be separated by a semi-colon (;), for example "opptyId;opptyType". When the hyper link is clicked, parameters `opptyId="oppty id value"` and `opptyType="type value"` will be passed as input parameters to the task flow.

```xml
<resource-view taskFlowId="/WEB-INF/OpportunityTF.xml#OpportunityTF">
  <parameters>
    <parameter name="viewId" value="Opportunity"/>
    <parameter name="webApp" value="CRMApp"/>
    <parameter name="pageParametersList" value=""/>
    <parameter name="taskParametersList" value=""/>
    <parameter name="resourceParamList" value="opptyId"/>
  </parameters>
</resource-view>
```

Custom navigation is handled by the Applications Core ResourceViewHandler registered in the `adf-config.xml` file. You must add this entry to all `adf-config.xml` files:

```xml
<wpsC:adf-service-config xmlns:wpsC="http://xmlns.oracle.com/webcenter/framework/service">
  <resource-handler
```

---

Implementing Activity Streams and Business Events  
Oracle Fusion Applications Developer's Guide
15.5.6 How to Implement Comments and Likes

Commenting allows users to comment on objects that are created or published by various users on the site, and engage in discussions revolving around those objects using replies to comments and comments upon comments. This feature in Activity Stream allows users to comment on a specific Activity related to a object.

The Likes feature allows users to express their liking for any object in the system to which they have access. This feature is exposed in message boards, Activity Streams, doclib and replies on topics in discussion forums. In Activity Stream, this feature allows users to indicate if they like a particular Activity.

To enable Comments and Likes for a service, add these Activity Types to the service-definitio.xml file:

```xml
<activity-type name="postComment"
    messageFormatKey="ACTIVITY_COMMENT_STATUS"/>
<activity-type name="expressLike"
    messageFormatKey="ACTIVITY_LIKE_STATUS"/>
```

Ensure the `activity-type` names are as shown. The `messageFormatKey` values refer to the `ResourceBundle` keys that provide strings displayed for "comments" and "likes" links displayed in the Activity message.

15.5.7 How to Implement Follow for an Object

Users will be able to see Activity messages belonging to the business objects they are following. Users should either explicitly follow a business object, or you should provide a way for users to follow certain business objects implicitly. A business object can be followed for a user by using the Oracle WebCenter Portal Follow API. A sample implementation of the Follow model is shown in Example 15-13.

Example 15-13  Sample Implementation of the Follow Model

```java
public void follow() {
    System.out.println("Follow method invoked!!");
    try {
        OAViewObjectImpl vo = getCaseList1();
        Row row = vo.getCurrentRow();
        Object id = row.getAttribute("Id");
        System.out.println("Case Id in follow : " + id);
    }
    catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
```
ActivityStreamingService asService = ActivityStreamingServiceFactory.getInstance().getActivityStreamingService();
FollowManager followManager = asService.getFollowManager();
String serviceID = "oracle.apps.fnd.applcore.crdemo.model.business.CasesEO";
String userGUID = ApplSessionUtil.getSession().getUserGuid();
ActivityActor actor = asService.createActor(userGUID);
String objectType = "Id";
ServiceObjectType objectType = asService.findObjectType(serviceID, objectType);
ActivityObject followedObject = asService.createObject(id.toString(), objectType, "");
System.out.println("Calling Follow for Case : " + id);
followedObject.setServiceID(serviceID);
followManager.followObject(actor, followedObject);
}
catch(ActivityException ae) {
    ae.printStackTrace();
    System.out.println("Case follow failed");
}

15.5.7.1 Defining the Service Category

The Follow model is enforced for Activity messages when the category-id of a service contains "business" in its name. A sample service-category-definition and its reference in service-definition are provided in Example 15-14 and Example 15-15.

Note that the ID in the service-category-definition file matches the category-id in the service-definition.xml file and it contains "business".

Example 15-14    Sample service-category-definition.xml File

<service-category-definition xmlns="http://xmlns.oracle.com/webcenter">
    <category id="oracle.apps.fnd.applcore.crdemo.model.business.CasesEO" resourceBundle="oracle.apps.fnd.applcore.crdemo.BusinessActivityServiceResourceBundle" titleKey="CASE_SERVICE_CATEGORY" icon="/a/s/g.gif"/>
</service-category-definition>

Example 15-15    Sample service-category-definition Reference

<category-id>oracle.apps.fnd.applcore.crdemo.model.business.CasesEO</category-id>

15.5.7.2 Adding ActivityTypes for Follow and Unfollow

The Activity types shown in Example 15-16 should be added to the service-definities of all services that use the Follow model. These Activity types are used to construct the message published when an object belonging to the service is Followed or Unfollowed.

Example 15-16    Adding ActivityTypes for Follow and Unfollow

<activity-type name="followObject" display="Follow Object" messageFormatKey="ACTIVITY_FOLLOW_OBJECT_MSG" description="Follow Object"/>
15.5.8 How to Render Contextual Actions in Activity Streams

Contextual Actions are rendered for business objects or other resources referenced in Activity Stream messages when `contextInfoPopupId` is configured in the `service-definition.xml` file of the business object or resource. The Activity Stream starts an Oracle ADF popup using the popup ID from the `service-definition.xml` file. The `contextInfoPopupId` should provide the absolute ID of the popup used for the Contextual Action. A popup with the specified ID should exist in the pages where the Activity Stream is used. This is a requirement for all pages where Contextual Actions-enabled objects are rendered. The Activity Stream will make the `serviceId`, `resourceId`, and `resourceType` properties available to the started popup. The popup should process these parameters and convert them to Contextual Actions-specific parameters and make them available to the Contextual Actions task flow or another component.

This element, which is the direct child of the `service-definition` element, is used to configure the Contextual Actions popup ID in the `service-definition.xml` file.

```xml
<contextInfoPopupId>:pt1:r1:casePopup</contextInfoPopupId>
```

The popup sample shown in Example 15-17 uses the `serviceId`, `resourceId`, and `resourceType` properties from the Activity Stream that are made available through the `launch` variable, and makes them available to the popup.

**Example 15-17 Sample Popup for ActivityStream**

```xml
<af:popup id="casePopup" contentDelivery="lazyUncached" eventContext="launcher" launcherVar="source" clientComponent="true">
    <af:noteWindow id="nw">
        <af:panelFormLayout id="pflTst">
            <af:inputText id="itSID" label="Service ID" value="#{pageFlowScope.serviceId}" readOnly="true"/>
            <af:inputText id="itRID" label="Resource ID" value="#{pageFlowScope.resourceId}" readOnly="true"/>
            <af:inputText id="itRType" label="Resource Type" value="#{pageFlowScope.resourceType}" readOnly="true"/>
        </af:panelFormLayout>
    </af:noteWindow>
    <af:setPropertyListener from="#{source.attributes.serviceId}" to="#{pageFlowScope.serviceId}" type="popupFetch"/>
    <af:setPropertyListener from="#{source.attributes.resourceId}" to="#{pageFlowScope.resourceId}" type="popupFetch"/>
    <af:setPropertyListener from="#{source.attributes.resourceType}" to="#{pageFlowScope.resourceType}"/>
</af:popup>
```
15.6 Implementing the Oracle Fusion Applications Search Results UI

The Oracle Fusion Applications Search Results UI is an implementation of the Oracle Enterprise Crawl and Search Framework (ECSF) within the Applications Core UIShell and SimpleUIShell (FUSE+) page templates. It allows all users on any page to perform searches in a consistent manner.

The minimum requirement is to implement and run a UI Shell template page. A page using the UI Shell template automatically will contain the search components in the Global Area and can be activated when running the page.

Where you have implemented ECSF for your product, you must ensure that you have followed all of the instructions in Set Up Your Development Environment, Getting Started with Oracle Enterprise Crawl and Search Framework, and Creating Searchable Objects and in particular:

- "How to Create a Database Connection" in Set Up Your Development Environment. The database connection must be defined inside the project using the UI Shell template. The Oracle Fusion Applications Search Results UI functionality uses the database connection SearchDB defined in this section to connect to the Oracle database to query the results.

- Creating Searchable Objects to make your view objects searchable. ECSF uses these search-enabled objects in the construction of the result set.

If you have implemented ECSF and defined the SearchDB connection, the saved and recent searches are saved to the Oracle database and are persisted across sessions.

Security on search is implemented by a combination of Functional Security over search categories at the ECSF layer, and Data Security (limiting search results to only those items to which the user has authorized access) using Applications Core data security rules on the searchable view objects.

15.6.1 How to Disable Oracle Fusion Applications Search

There are occasions when you will want to disable Oracle Fusion Applications Search for an application, such as for public (unauthenticated) pages. There are four ways to disable the function:

- The preferred method is to set the profile option Fusion Apps Search Enabled to 'N', either at site level or user level.

- Setup switch using a JVM system property.
  -DFUSION_APPS_SEARCH_ENGINE_AVAILABLE=N

- Use Customization by setting rendered to false on the panelGroupLayout with id "_UISGlobalSearch" (the panel containing the Oracle Fusion Applications Search fields) in the UI Shell main area. Note that by customizing the fields from the current page, you are not disabling search; the Expression Language bindings on the fields are still evaluated and if the user opens a non-customized page, Oracle Fusion Applications Search will be available.

- Remove the ECSF libraries (oracle.ecsf shared lib). Oracle Fusion Applications Search will detect the missing dependency and disable itself for all
pages in the current user session, even if the user opens a web application that does have the ECSF libraries available.

15.6.2 How to Use Basic Search

From the main page of the project in the Global Area, click the Search icon to display the Search dialog:

**Figure 15-21  Initial Search Dialog**

When you click the search icon, a popup will show suggestions. By default, these will contain any Recent Searches, Saved Searches and Recent Items the user has performed. As the user types, these suggestions will narrow down, and other suggestions, such as previous search terms, will be shown.

The Groups of suggestions shown to the user are determined by the search configuration in effect (see Using Setup Task Flows in), and personalizations made by the user.

By clicking the gear icon, the user can access these personalizations and can select/deselect suggestion groups, reorder them, and so on.

When you click **OK**, the selections are saved as MDS personalizations that will be stored across login sessions for the user.

**Hint:** To reset the personalizations, hold down the **Ctrl** key and click the gear icon. Your previous selections will be deleted and you will see the default configuration the next time you log in.
When you click a Recent Searches link or a Saved Searches link, the search will be run. When you click a Recent Items link, the task flow will be re-opened. As you type in the search field, the characters typed will narrow down the suggestions shown for the Recent Items, Recent Searches and Saved Searches. In addition, suggestions for previous searches will be shown.

**Recent Searches**

When you start to type in the search field, the Recent Searches list will refresh and narrow down based on the substring you type.

Keywords entered by the user serve as the name of the recent search.

Recent searches are implemented using the ECSF recent searches feature. See [About Managing Recent Searches](#).

A recent search is uniquely identified by its filters, such as search term, categories, and facet selections.

A search will be added to the front of the recent search list when performed. If it already exists in the list, it will be moved from its current place in the list to the front.

**Search Term**

This is a text field for the values on which to search. The field can hold a maximum of 2048 characters.

The term is searched for in any of the crawled data, which includes the title, fixed and variable content, attached documents, and tags. So if the search term is foo, the search returns any data containing the word foo.

Click the **Play** button or press **Enter** to initiate the search.

Click the **Show Filters** link to view and change the filters that will be used for the search.
Once a search has been performed, the Search terms field and the search results are displayed, as shown in Figure 15-24.

Sort

If multiple results are displayed, the Sort icon to the right of the More than X Results heading becomes active. Click the icon to display the Sort By dialog.

This function requires no developer implementation; it is built in.

Users can sort results in the results table. The sort may be done in ascending or descending order, and may be switched using a toggle button.

The sort will be available in two forms, depending on the search parameters:

- Multiple categories or a single category
The sort can be based only on Relevance (an implicit universal attribute) and LastModifiedDate (a universal attribute).

- A Single Searchable Object in a category
  The sort also can be based on Relevance and LastModifiedDate, and any stored attributes defined on the searchable object.

The Search result will be expanded in the background from the initial 10 results returned with the query, to 100 results returned by a background search started in a separate thread as soon as the 10 results are successfully returned. This is to give a reasonable result to sort. The sort UI will show a spinner and will be disabled until this is finished, and the UI will poll for completion every two seconds and enable those fields. This polling will stop when the background search is complete, or after a fixed number of polls (to stop infinite polling in case of error).

The default Sort is Relevance descending, which is the way that records are returned by SES.

The sort is done in memory using a standard Java Collections.sort function, and a comparator that takes into account the date type of the attribute (Date/Number/String) and direction.

15.6.2.1 Using the Search Filters

Click **Show Filters** to display them on the left side of the display. They are sized to make them easy to use on touch-screen appliances.

Last Updated Date and Categories are default filters.

*Figure 15-26  Search Filters Buttons*

**Last Updated Date**

Click **Last Updated Date** to display the selections panel, similar to *Figure 15-27*. 
The valid values are:

- **All**
  
  Clears the current filter and displays results for all available dates.

- **Today**
  
  LastModifiedDate equals "todays date."

- **This Week**
  
  LastModifiedDate >= "last Sunday" AND LastModifiedDate <= "this Saturday"

- **This Month**
  
  LastModifiedDate >= "first day of month" AND LastModifiedDate <= "last day of month"

- **This Year**
  
  LastModifiedDate >= "first day of year" AND LastModifiedDate <= "last day of year"

- **Custom Date Range**
  
  As appropriate for the range. A date picker will be displayed. The dates in the custom date range default to the last year (last 365 days).

### Categories

**Note:**

Categories are not set up at design time by developers. They should be set up either by customers or seeded by teams. Categories are created and stored in ECSF schema, and ECSF provides an API to get a list of categories to the UI for a given user. See How to Control the Look and Feel of Search.

When you click **Categories**, a list, similar to that shown in Figure 15-28, is displayed:
The user can select from the list of Categories. Unchecking the All category unchecks all of the categories. The subset of selected categories will be displayed in the entry area of the dropdown list as a concatenated list separated by commas.

Alternate Word List

Oracle Secure Enterprise Search (SES) will show alternate words to the user when they do a search as suggestions to frequent typos, or better used terms. This list is based on statically defined lists stored in SES.

Oracle Fusion Applications Search uses the ECSF APIs to show the alternate words to the user. Clicking the alternate word does a new search using the selected alternate word as the new keyword.

Saved Searches

Click Save to open a list of saved searches.

To create a new saved search that uses the last search string seen in the Search window as the search criteria, enter a name in the Name field and click OK. Make sure you have not selected an existing search.

To rename a saved search, select it, enter a new name in the Name field, and click OK.

To delete a saved search, select it and click Delete.

15.6.2.2 Search Results

After clicking Search, a modal dialog will display the results of the search. Hovering over the main link will show the last crawled date. Figure 15-31 shows typical results.
Note:

If a search application, such as Finance or HCM, does not respond to the search request within a predetermined period, the search results will be displayed but there will be a notice that one or more applications did not respond.

Figure 15-31  Search Results Example

The Search Results display consists of:

- A repetition of the fields displayed in the Global Area.
- A set of filter tiles: Selected filter values will be applied to the search results.

A category is a group of related objects. Examples include any Oracle Fusion business object, and Oracle WebCenter Portal objects such as wikis and blogs.

A Searchable Object is the second level. A searchable object is the view object.

If a category has only one Searchable Object, it is not shown as a filter because there is no action a user can take on it; it is automatically selected. Figure 15-31 shows an "Emp Cat" that has one searchable Object (not shown) that has three facet filters.

Facets are formed by the Lists of Values defined on an attribute in the Searchable Object. There may be many facets for a Searchable Object, and the facets may be hierarchical, such as is the case in Figure 15-31, where a State facet contains a County facet, which contains a City facet. Only the name of the highest level facet in the hierarchy is shown.

- A Results section. The search results use the keywords in the search field with the filters applied.

Each result found under a category provides a navigation link back to the record.

The format of a search result is:
- A line containing the default or primary action for the result. This may include an icon if it is defined on the search action. This line will be seen for all search results. Clicking the link navigates the user to the primary destination for the result.

- A line containing the fixed content of the result. If any search terms match in this text, they are shown in bold. The Fixed Content is derived from the Search Properties Title field as defined in ECSF.

- A line containing the variable content of the result. If any search terms match in this text, they are shown in bold. This text will wrap if it overflows the width of the line. The Fixed Content is derived from the Search Properties Body field as defined in ECSF.

- A line of tags attached to this result of the form "Tags: tag1 tag2 ...". If there are no tags, this line is not shown.

- A line of other actions in alphabetic order. Other actions are similar to the default action in that they may consist of an icon followed by a link. Clicking the link navigates the user to subsidiary destinations for the result. For example, the primary action for a result on a purchase order search may be to open the found purchase order. Other actions may include approval or rejection of the purchase order.

- A line of attachments for this result of the form "Attachments: attach1 attach2 ...". Each attachment may consist of an icon followed by the attachment name. A search may match the search term within the attachment. Clicking an attachment will download and open the attachment. If there are no attachments, this line is not shown.

**Result Counts Do Not Add Up**

It is possible when viewing the search results, and narrowing your selections using the facet tree, to see counts against the nodes that do not add up. For example, a search on Glasses might return 16. Then if you filter the result by color, you may find Blue (5) and Red (10), which do not add up to 16. This count is the Oracle Secure Enterprise Search (SES) Approximate count based on heuristics, and not an exact count. To make the count exact, start SES and select Global Settings > Query Configuration, and click the Exact count radio button, as shown in Figure 15-32. Note that SES warns against this for performance reasons. See the Oracle Secure Enterprise Search Administration Online Help.
15.6.3 How to Control the Look and Feel of Search

15.6.3.1 Using the Setup Task Flows

Two setup task flows are available for developers and customers to use in extending the Global Search metadata.

They will be available in the following Applications Core Libraries.

Table 15-8  Exported JAR Files

<table>
<thead>
<tr>
<th>JAR</th>
<th>Level</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>GlobalSearch-Model.jar</td>
<td>Model</td>
<td>Applications Core (oracle.applcore.model.ea)</td>
</tr>
<tr>
<td>GlobalSearch-View.jar</td>
<td>ViewController</td>
<td>Applications Core (Setup UI) (oracle.applcore.setupui.war)</td>
</tr>
</tbody>
</table>

The created task flows are:

Table 15-9  Created Task Flows

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow Id</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Global Search Configurations</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/globalSearch/uiPublic/GlobalSearchManageConfigurationsTF.xml#GlobalSearchManageConfigurations</td>
<td>Set up and manage the options passed to Global Search on a per View basis.</td>
</tr>
</tbody>
</table>
### Table 15-9  (Cont.) Created Task Flows

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow Id</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Global Search Suggestion</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/globalSearch/uiPublic/GlobalSearchManageGroupsTF.xml#GlobalSearchManageGroups</td>
<td>Set up and manage Search Suggest Groups.</td>
</tr>
</tbody>
</table>

#### 15.6.3.2 Using Setup Task Flows in Fusion Applications

To access the setup flows in Fusion Applications:

1. Select **Setup and Maintenance** from the Settings and Actions dropdown on the top of the page. See [Using the Settings and Actions Menu](#).

2. On the **Overview** page, select the **All Tasks** tab.

3. Set the **Search** dropdown to "Task Lists and Tasks."

4. Enter "global search" in the **Name** field and click **Search**.

The Application Global Search Configuration and Manage Suggestion Groups task flows are available under Define Global Search:

![Figure 15-33 Searching for Define Global Search in Settings and Actions](image)

Select the task flow you want to manage and click the Go to Task icon. The details are described in [Setting Up a Global Search Configuration](#).

#### 15.6.3.3 How a Configuration Lookup Is Processed

The configuration is loaded for the page each time it is loaded.

The algorithm is as follows:

- Determine the `viewId` and `application` for the loading page.
- Search for a configuration of type `Page` that matches the current `viewId` (unSEEDED, then SEEDED). If found, use it.
• Search for a configuration of type Application that matches the current application (unSEEDED, then SEEDED). If found, use it.

• Load the default.

In all cases, "like" searches are done, so SQL wildcards can be used and are recommended. For Application, these are required at the end because the application name often comes back with a version number, such as "HomePageApp(2.0)", so use an expression such as "HomePageApp%".

In all cases, if there is a customer (non SEEDED) configuration, it will be loaded before the SEEDED version.

15.6.3.4 Set Up a Global Search Configuration

As a developer there is no need to set up and ship a configuration to control the look and feel of Global Search. There is a default configuration that ships and Global Search will behave exactly how it always has. However, if you wish to change how Global Search appears to your customers, you can make and ship your own configuration. How this affects the search UI is described in this section.

Once you have clicked the Go to Task icon, described above, this dialog is opened:

Figure 15-34    Manage Global Search Configurations

From 1 (Short Name) to 11 (Last Updated Date), the fields are:

• (1) **Short Name**: The short name of the configuration. This is not shown in the Global Search UI.

• (2) **Display Name**: The display name of the Configuration. This is not shown in the Global Search UI. This value is translated.

• (3) **Description**: The Description of the Configuration. This is not shown in the Global Search UI. This value is translated.

• (4) **Default**: Is this the default configuration. There can be two Default configurations - a SEEDED configuration supplied by applcore that is the default configuration of Global Search. It cannot be changed or edited (the UI will prevent this), and a single default designated by the customer. A customer default will be loaded before the SEEDED default.

No Oracle group should create a default group and ship it as SEED data. This is not shown in the Global Search UI.

• (5) **Enabled**: Is the group enabled. Only enabled groups can be used. This is not shown in the Global Search UI.

• (6) **Product Family**: The product family of the configuration. A configuration can only configure search groups in the Common product family (shipped by applcore and available on every pillar) and the same Product Family. This ensures that the Suggestion Group class can be loaded at runtime.
This is not shown in the Global Search UI.

- **Module**: The module of the Configuration. This is used by the SEED data extraction process to distinguish between different groups’ SEED data. This is not shown in the Global Search UI.

- **Created By**: Row Who. This is not shown in the Global Search UI. A row that has Created By = SEED_DATA_FROM_APPLICATION is considered SEED data.

- **Creation Date**: Row Who. This is not shown in the Global Search UI.

- **Last Updated By**: Row Who. This is not shown in the Global Search UI.

- **Last Updated Date**: Row Who. This is not shown in the Global Search UI.

### 15.6.3.5 Edit Global Search Configuration

To edit a global search configuration, follow these steps:

1. Select the configuration, as shown in Figure 15-34, and click the Edit icon.

   **Note**: SEEDED configurations cannot be edited. In particular, the SEEDED DEFAULT configuration can never be edited by developers or end users.

   This dialog, with the Autosuggest tab selected, is shown:
The fields in the top portion are described with Figure 15-34.

From 12 (Enable personalization ...) to 19 (Maximum Number ...), the fields are:

- **(12) Enable personalization of search groups**: Is the tab of Suggest groups seen in the search field popup behind the gear icon?  
  If this and "Enable personalization of search categories" are no set, the gear icon is not shown.

- **(13) Suggestion Groups**: The set of Suggestion groups to show. This is an initial developer design decision and determines the groups the user sees in the search popup. The user then can select a subset of these and this selection
will be remembered in MDS. Shuttle the groups across that you want to show and override the default enabled state using the picker.

- (14) **Show Suggestion Group Headings**: Whether to show the suggestion group header. The header includes an icon (if available) and the heading text. Be aware that turning headings off may run suggestions from different groups together making it hard to distinguish them.

- (15) **Show Icons**: Should the suggestions show icons? If there is no icon available, or this is set to no, a bullet is shown.

- (16) **No Suggestions Message**: Message to show if there are no matching suggestions. If there are no matching suggestions, the suggestion popup will not be shown.

- (17) **Show Top Suggestions**: Whether to show top suggestions. Top suggestions are those shown in the suggestion list before the user has entered the “Minimum Characters for Autosuggest” and typically contains a static list of highly used suggestions as desired by the suggestion group author.

- (18) **Minimum Characters for Autosuggest**: The minimum number of characters in the search field before search suggest switches from showing top suggestions to suggestions specific to the typed values.

- (19) **Maximum Number of Suggestions**: The total maximum number of suggestions shown in the suggest list, spread evenly across all groups.

2. When finished editing the Autosuggest features, click the Search Field tab to display these settings:

![Figure 15-36 Edit Global Search Search Field Configuration](image)

From 20 (Minimum Number ...) to 24 (Placeholder), the fields are:

- (20) **Minimum Number of Characters**: The Minimum number of characters required for searching. By default this is 1. This will affect the error message “You must enter keywords to search”, “You must enter x character(s) to search.”

- (21) **Maximum Number of Characters**: The maximum number of characters the search field will allow. This sets the underlying af:inputText columns attribute.

- (22) **Automatically Clear Field Value after Search**: Is the search field cleared after the search. This is a convenience allowing the user to enter new text without having to clear or edit the previous search terms.
• **Label**: The Label of the Search field. To have no label, leave this value blank. For accessibility purposes, the hidden label value will be "Search."

• **Placeholder**: The placeholder or shadow text of the Search field. To have no placeholder, leave this value blank.

3. When finished editing the Search Field features, click the Search Results tab to display these settings:

**Figure 15-37  Edit Search Results Configuration**

![Image of Edit Search Results Configuration](image)

From 25 (Enable saved searches) to 36 (Show Icons), the fields are:

• **Enable saved searches**: Whether the user can save a search. The **Save** button will be hidden.

• **Enable recent searches**: Whether recent searches are saved automatically as the user runs and filters searches.

• **Filter Display Style**: Whether the filters (Category filters only) are shown inline or using a popup.
(28) **Enable personalization of search categories**: Is the tab of Search Categories seen in the search field popup behind the gear icon? If this and "Enable personalization of suggest groups" are not set, the gear icon is not shown.

(29) **Show subcategories in the search results**: Should Searchable Objects, also called subcategories, be shown? If they are not shown, "Show Facets" is also false.

(30) **Show Facets**: Should facets for the selected subcategory be shown?

(31) **Show hit counts**: Should hit counts in the facets be shown?

(32) **Enable clear all filters**: Should the *Clear Filters* button be shown?

(33) **Application**: The application to which to restrict the search categories. This is optional and will restrict the categories shuttle to only the categories from that application. This drop box is populated with the unique applications with which all categories are registered in the ECSF database.

(34) **Categories**: The set of categories to show. This is an initial developer design decision and determines the categories the user sees in the search popup. The user then can select a subset of these and this selection will be remembered in MDS. To pick all categories (including those added in the future), leave the categories unshuttled.

(35) **Last Updated Date Filters**: The set of Last Updated Date to show. To pick all filters (including those added in the future), leave the filters unshuttled.

(36) **Show Icons**: Whether icons are shown in the results.

4. When finished editing the Search Results features, click the Pages tab to display these settings:

*Figure 15-38  Edit Global Search Pages Configuration*

The two fields are:

(37) **View Type**: The View Type to which to map this configuration. There are two types of view type: Application and Page. You may enter as many rows as needed here.

(38) **View ID**: This value depends on the View Type:

- **Application**: Enter the application short name with a wildcard at the end, for example HomePageApp%.

  To find the application name:
  1. Open the Setup and Maintenance work area and go to the Manage Taxonomy Hierarchy task.
2. Expand the Oracle Fusion node, select a row with the Application module type, and click **Edit Module**.

3. In the Application Details section, see the Application Short Name column for the value to use. 

   **Page**: Enter the last part of the URL you get when you open that page. For example, enter ExamplePage from the URL.

   ```
   http://exampleServer/homePage/faces/ExamplePage
   ```

   Note that you cannot enter blind expressions, such as "%" because this effectively short cuts the configuration lookup algorithm. You **must** enter at least 5 characters that are **not** wildcard characters.

5. Click **Save and Close** or click the Manage Suggest Groups Configuration tab if it is available at the top of the page.

### 15.6.3.6 Manage Suggestion Groups Configuration

If Suggestion Groups have been implemented (see Create Suggestion Groups, the Manage Suggest Groups Configuration tab will be made available next to the Search Configurations tab, as shown here:

![Manage Suggestion Groups Configuration](image)

To manage a Suggestion Group, follow these steps:

1. Select a group and click the Edit icon. This configuration dialog is displayed:

   ![Edit Suggestion Group](image)
From 1 (Short Name) to 16 (Icon), the fields are:

- **(1) Short Name**: The Short name of the Suggestion Group. This is not shown in the Global Search UI.
- **(2) Display Name**: The Display name of the Suggestion Group. This is not shown in the Global Search UI. This value is translated.
- **(3) Description**: The Description of the Suggestion Group. This is not shown in the Global Search UI. This value is translated.
- **(4) Displayed by default**: Is this group displayed by default? This determines whether the suggestion group (see item 13 for Figure 15-35) is enabled by default. If it is not, the developer or customer can override the setting to enable the group. This allows a group to be registered and shipped in different configurations with different enabled items.
- **(5) Product Family**: The product family of the suggestion group. A Global Search Configuration can only configure search groups in the Common product family (shipped by applcore and available on every pillar) and the same Product Family as the group is declared in. This ensures that the Suggestion Group class can be loaded at runtime by configurations in effect. This is not shown in the Global Search UI.
- **(6) Module**: The Module of the Configuration. This is used by the SEED data extraction process to distinguish between different groups' SEED data. This is not shown in the Global Search UI.
- **(7) Created By**: Row Who - this is not shown in the Global Search UI. A row that has "Created By" = SEED_DATA_FROM_APPLICATION is considered SEED data.
- **(8) Creation Date**: Row Who. This is not shown in the Global Search UI.
- **(9) Last Updated By**: Row Who. This is not shown in the Global Search UI.
- **(10) Last Updated Date**: Row Who. This is not shown in the Global Search UI.
- **(11) Data Source**: The suggestion group. This can be:
  - One of the predefined suggestion groups. These are supplied in an autosuggest list on the field.
- **(12) Context Code**: The Context Code for the suggest group, if it is known at registration time and is to be used in Global Search. Different group use contextCode as an initialization value in different ways. For example the Recent Items group will strip items by the contextCode and objectType. By default, Global Search sets no contextCode and objectType.
- **(13) Object Type**: The Object Type for the suggest group; a sub-classification of Context Code, if it is known at registration time and is to be used in Global
Search. Different groups use `objectType` as an initialization value in different ways. For example, the Recent Items group will strip items by the `contextCode` and `objectType`. By default, Global Search sets no `contextCode` and `objectType`.

- **(14) Heading Visible:** Should the heading be shown for this group? Note that heading can be turned off for all groups at the "Global Search Configuration" level (see call out 14 in that section) overriding this value, though if the heading is off here it cannot be turned on at all.

  Be aware that turning headings off may run suggestions from different groups together and make it hard to distinguish them.

- **(15) Text:** The Heading Text for the group. If set will override the heading text set in the java code of the Suggestion Group.

- **(16) Icon:** The Heading Icon for the group. If set will override the icon set in the java code of the Suggestion Group.

2. Click **Save and Close**.

### 15.6.3.7 Create Suggestion Groups

Developers can create a suggestion group by extending the `oracle.apps.fnd.applicore.patterns.ui.behaviors.searchSuggest.AbstractSearchSuggestGroup` class.

The simplest way to test at this stage in development is to attach it to an `af:inputText` field in a simple UI:

```xml
<af:inputText label="My Test Suggest Group: " id="simple">
  <fnd:searchSuggestBehavior suggestGroup1="oracle.apps.tla...MySuggestGroup" contextCode1="ifNeeded" objectType1="ifNeeded"/>
</af:inputText>
```

To create, register and ship the suggestion group, follow these steps:

1. Use this boilerplate example to create a suggestion group. You will need to fill in the TODO items.

```java
import oracle.apps.fnd.applicore.patterns.ui.behaviors.searchSuggest.AbstractSearchSuggestGroup;
import oracle.apps.fnd.applicore.patterns.ui.behaviors.searchSuggest.SearchSuggestItem;

public class RequiresSubmitSearchSuggestGroup extends AbstractSearchSuggestGroup {
  /**
   * Unique name of this group. Must be unique across all groups.
   */
  public static final String NAME = "uniquegroupname";
  // TODO add NAME.
  @Override
  public String getName() {
    return NAME;
  }
}
```
@Override
public String getDisplayName()
{
    // Allow override of Heading in Manage Suggest Groups UI.
    String overRiddenDisplayName = super.getDisplayName();
    if (overRiddenDisplayName != null)
    {
        return overRiddenDisplayName;
    }
    return "Translated Heading";
    // TODO add translated heading.
}

@Override
public String getIconURL()
{
    // Allow override of Icon in Manage Suggest Groups UI.
    String overRiddenIconURL = super.getIconURL();
    if (overRiddenIconURL != null)
    {
        return overRiddenIconURL;
    }
    return "some/path/to/image.png";
    // TODO add icon.
}

@Override
public List<SearchSuggestItem> getItems(String searchText,
    int maxSuggestedItems)
{
    List<SearchSuggestItem> ret = new ArrayList<SearchSuggestItem>();
    // TODO add suggestion items based on searchText.
    // This call MUST return as fast as possible.
    ret.add(...);
    return ret;
}

@Override
public List<SearchSuggestItem> getTopSuggestions(String searchText,
    int maxSuggestedItems)
{
    List<SearchSuggestItem> ret = new ArrayList<SearchSuggestItem>();
    // TODO add top items.
    // This call MUST return as fast as possible.
    ret.add(...);
    return ret;
}

@Override
public void init(Map<String, Object> attributes)
{
    // TODO: Look-up suggestions to use in getItems / getTopSuggestions if needed
to allow fast calls of getItems / getTopSuggestions.
    // If they need to be looked up during those calls, you MUST ensure the
lookup is fast as it will block the suggest field.
}
```java
@Override
public boolean requiresSubmit()
{
    // TODO: true if the group value selection needs processSubmit(..) to process
    // the selection, false if the suggestions are keywords
    return true;
}

@Override
public boolean placeSelectedItemInField()
{
    // TODO: true to place the selected item in the search field, false
    // otherwise.
    return true;
}

@Override
public void processSubmit(String item, Map<String, Object> attributes)
{
    // TODO: Write code to look-up item and process it according to business
    // rules.
}
}
```

2. Register the group in the setup UI.

If your suggestion group is to be used in your own UI, just add it to your code.

If the group is to be used in Global Search, register it in the "Manage Suggestion Group" UI. Use the fully qualified class name as the "Data Source". Use your "Product Family" and "Module".

Set a "Context Code" and "Object Type" if necessary.

3. Register a configuration and include your group in it.

Create a new configuration, or re-use an existing one that you control. The "Product Family" and "Module" should be the same as for the group you have registered.

Set options as appropriate for your use case. Include the new group and any Common (Applications Core defined) groups you need. Override the display status using the "Enabled" drop box if you wish to change it.

Map the configuration to your UI pages and applications.

Do not set the configuration as a default configuration. There should be only one default configuration and that is developed and shipped by applcore. A customer can set a configuration of their own as a default to override the Applications Core default, but it should not be Seeded.

4. Extract and ship the setup.

Development teams will:

- Use the setup UI to define their metadata.
- Ensure their module is set correctly for their LBA.
- Use standard Seed data extraction utilities to extract that metadata and ship in their patches.

Seed data will be extracted separately for:
Teams will only extract and ship Seed data for the modules they own.

Use commands similar to these:

```
-AM oracle.apps.fnd.applicore.search.model.applicationModule.FusionSearchAM \
-VO FndFusionSuggestGroups \
-ExtractRootPath $ADE_VIEW_ROOT/atgpf/applcore/db/data
```

Use the

```
[-PartitionKeyIds <Taxonomy ModuleId values> | -PartitionKeyNames <Taxonomy short name values>]
```

parameter to get the Seed data for your product. See Initializing Oracle Fusion Application Data Using the Seed Data Loader for a full description of Seed Data.

### 15.6.4 Introduction to the Crawled Objects Project

The Crawled Objects Project lets you crawl your Search view objects in Oracle WebLogic Server, and set up Oracle Fusion Applications Search to use those crawled view objects.

The business component objects you will create (specifically the Searchable view object) will contain references to the Oracle Fusion Middleware Extensions for Applications base classes.

Update the ECSF command-line script (`runCmdLinScript.sh`) to reference the JAR files containing the base classes. Example 15-18 shows the UNIX version; the DOS version is similar.

#### Example 15-18 Updating ECSF Command-Line Script to Reference Applications Core JAR Files

Run these commands if you are using a Linux operating system:

```
export APPLCORE_CP=${ORACLE_HOME}/jdeveloper/jdev/oaext/adflib/Common-Model.jar:$ORACLE_HOME/jdeveloper/jdev/oaext/adflib/Tags-Model.jar
export ADMIN_CLASS=oracle.ecsf.cmdlineadmin.CmdLineAdmin

${JAVA_HOME}/java -cp ${ADMIN_CP}:${APPLCORE_CP} ${ADMIN_CLASS} ${CONNECT_INFO}
```

Run these commands if you are using a Windows operating system:
set APPLCORE_CP=%ORACLE_HOME%\jdeveloper\jdev\oaext\adflib\Common-Model.jar;
%ORACLE_HOME%\jdeveloper\jdev\oaext\adflib\Tags-Model.jar
set ADMIN_CLASS=oracle.ecsf.cmdlineadmin.CmdLineAdmin

%JAVA_HOME%\java -cp %ADMIN_CP%;%APPLCORE_CP% %ADMIN_CLASS% %CONNECT_INFO%

15.6.5 How to Implement Tags in Oracle Fusion Applications Search

A view object is available to reference Oracle WebCenter Portal Tags. This view object is available in the ORACLE_HOME/jdeveloper/jdev/oaext/adflib/Tags-Model.jar library JAR file.

You may use this view object using a view link and a predefined Search extension to enable the crawling of Oracle WebCenter Portal Tags, both in initial and incremental (someone has updated the tags) crawls.

Follow these steps:

1. Create your Searchable view object as usual. Example 15-19 uses a Searchable view object over FND_LOOKUPS_VL in the query.

   The SERVICE_ID specifically identifies your Searchable view object.

   The RESOURCE_ID identifies the specific row for the SERVICE_ID. It is a dot-separated primary key of the entity.

   These two values will be used when setting up tags in your regular UI and must match. For example, if you have a page with a form and a tag button, the Oracle WebCenter Portal tag would be set up as shown in Example 15-20.

   **Note:**

   Do not forget to mark your key columns and ensure the order is consistent between the view object and the tag:taggingButton.resourceId attribute.

---

**Figure 15-41  Tags - Searchable View Object Lookup Types**

Figure 15-42 shows the attributes for Searchable view object lookup types.
2. Add a view link to the TagSVO (service view object) linking the Search view object and the Applications Core Tag view object. The view link should look similar to Figure 15-43.

3. Update the Body field to include the tags of the child view object in the relevant position in the string as defined by your management.
This will be an expression of the form `<accessor Name>.Tag`, such as `tagSVO.Tag`.

**How to Do an Incremental Crawl**

To do an incremental crawl:

1. Update the Searchable View Object Search Plugin field (see Figure 15-43) to "oracle.apps.fnd.applcore.search.TagSearchPlugin", or as shown in Example 15-21, create a subclass so that you can incorporate your security rules.

Ensure you add a parameter passing the service ID of the Search view object. This may be done by clicking the LOV symbol next to the Search Plugin field, shown in Figure 15-42. See Figure 15-44.

**Figure 15-44  Search Plugin**

![Search Plugin](image)

Note that the Plugin Class Name is the same as that described in Adding Attachments Support to Global Search.

There are two parameters, shown in Table 15-10, that may be passed to the extension.

**Table 15-10  Parameters that can be passed to the plug-in**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAG_SERVICE_ID</td>
<td>Yes</td>
<td>Service ID of the Searchable view object. This value must match the value in the tag:taggingButton component and the service_id of the Searchable view object.</td>
</tr>
</tbody>
</table>
Table 15-10  (Cont.) Parameters that can be passed to the plug-in

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_SPLITTER_CLASS</td>
<td>No</td>
<td>An optional class that extends oracle.apps.fnd.applcore.search.BaseKeySplitter. This is a strategy class for splitting the resourceId value into individual primary key attribute values. By default, oracle.apps.fnd.applcore.search.DefaultKeySplitter is used, which will split values based on a period separator (the applications standard). The separated PS attribute values are matched to the primary key columns in the order the primary key columns are defined in the view object flat table editor. For more flexible arrangements, teams can implement any scheme they want (such as name-value pairs) by creating their own key splitter class and setting this parameter.</td>
</tr>
</tbody>
</table>

2. Start the SearchFeedServlet in the user interface project.

Example 15-19  Creating a Searchable View Object

```
SELECT LOOKUP_TYPE,
    VIEW_APPLICATION_ID,
    LANGUAGE,
    SOURCE_LANG,
    MEANING,
    DESCRIPTION,
    CREATED_BY,
    CREATION_DATE,
    LAST_UPDATED_BY,
    LAST_UPDATE_DATE,
    LAST_UPDATE_LOGIN,
    'oracle.apps.fnd.applcore.lookuptype' AS SERVICE_ID,
    lookup_type||'.'||to_char(view_application_id)||'.'||language||'.'||meaning
as RESOURCE_ID
FROM FND_LOOKUP_TYPES_TL
```

Example 15-20  Setting up the Oracle WebCenter Portal Tag

```
<af:panelFormLayout id="pf11">
    <tag:taggingButton serviceId="oracle.apps.fnd.applcore.lookuptype" resourceName="#{bindings.Description.inputValue}" resourceID="#{(bindings.LookUpType.inputValue).#(bindings.ViewApplicationId.inputValue).#(bindings.Language.inputValue).#(bindings.Meaning.inputValue)}/">
        <af:region value="#{bindings.tagginglaunchdialog1.regionModel}" id="r1"/>
        <af:panelLabelAndMessage label="#{bindings.LookUpType.hints.label}" id="plam4">
            <af:outputText value="#{bindings.LookUpType.inputValue}" id="ot9"/>
        </af:panelLabelAndMessage>
    </tag:taggingButton>
</af:panelFormLayout>
```
**Example 15-21  Creating a Subclass**

```java
package oracle.apps.fnd;
import oracle.apps.fnd.applcore.search.TagSearchPlugin;
public class WlsTestTagSearchPlugin
extends TagSearchPlugin
{
    // All implementation through super class, or override methods important to you.
    // Be careful if implementing
    //  public Iterator getChangeList(SearchContext ctx, String changeType)
    // to call super(ctx, changeType) to get the applcore functionality.
}
```

You now can crawl using the command-line script. A full crawl will be done first, then on subsequent crawls the incremental functionality will call the `getChangeList()` method.

**15.6.6 How to Use the Actionable Results API with Oracle Fusion Applications Search**

This section details how to set up ECSF searchable objects to use with Oracle Fusion Applications Search. For information about setting up your global search infrastructure, see *Getting Started with Oracle Enterprise Crawl and Search Framework.*

*Figure 15-45* shows the result that will be produced (a single row in the search results table).

*Figure 15-45  Search Results Example*

<table>
<thead>
<tr>
<th>Flat Table URL Action</th>
<th>Title for Flat Table 1:Col1619:Col2619</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Include for Flat Table 1 with extra information to form a description: Col1619: Information for Col4: Col144: Any other required information would be added later</td>
</tr>
</tbody>
</table>

The terminology referred to in this result is:

- **Flat Table URL Action** is the Action Link.
- **Title for Flat Table 1:Col1619:Col2619** is the Fixed Content.
- **Any other required information would be added later** is the Variable Content.
- **Task Action 1** is Other Actions.

As shown in *Figure 15-46*, ECSF searchable objects support two distinct Action Types: URL and Task. See also *Figure 15-47* and *Figure 15-48*.

Most Oracle Fusion Applications will use the Task type with specific named parameters to integrate with the UI Shell; however both types will work.
Fixed Content
The Fixed Content is derived from the Search Properties Title field.

Variable Content
The Variable Content is derived from the Search Properties Body field.

15.6.6.1 Implementing the URL Action Type
Figure 15-47 shows a URL Action.
For URL Action Types, the Oracle Fusion Applications Search will open a new browser tab or window containing the URL. To configure this type, add a URL Search Result Action with these parameters.

- A unique name
- Action Type of URL
- An Action Target to the required destination, including groovy substitution parameters
- A Title

No Parameters are required; however a single iconURL parameter may be defined if an icon is required for the URL action. See Table 15-11.

### Table 15-11  iconURL Parameter

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iconURL</td>
<td>No</td>
<td>URL of icon to show next to the Action. Can be a relative reference such as /media/search/mime_doc.gif or a full URL such as <a href="http://host:port/path/to/icon.gif">http://host:port/path/to/icon.gif</a>.</td>
</tr>
</tbody>
</table>

The icon will be shown in the search results with the title given in the Title field. When clicked, a new browser tab or window will open with this URL.

### 15.6.6.2 Implementing the Task Action Type

**Figure 15-48** shows a Task Action.
For Action Types of Task, the Oracle Fusion Applications Search will open a UI Shell tab in the current page, or a new page containing the task flow. To configure this type, add a Task Search Result Action with these parameters:

- A unique name
- Action Type of Task
- A Title

Parameters are shown in Table 15-12. Note that, although this table resembles Table 15-13, it presents the use case that the majority of users will use. The information in Table 15-13 is for a very small use case.

The action will be shown in the search results with the title given in the Title field. When clicked, a new UI Shell tab window will open with this task flow. If the viewId parameter is for the current page, the UI Shell tab will be in the current page; otherwise the current page will be replaced with a new UI Shell page with the search result.

---

**Note:**

Do not enter double quotation marks around the groovy expressions; use single quotation marks instead.

---

**Table 15-12  Task Action Type Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewId</td>
<td>Y</td>
<td>Name of the page. This is shown in the browser URL bar. For example, in <code>http://127.0.0.1:8989/context-root/faces/TestUIShellPage</code>, it would be <code>/TestUIShellPage</code> (Note the leading slash).</td>
</tr>
</tbody>
</table>
Table 15-12  (Cont.) Task Action Type Parameters

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pageParametersList</td>
<td>N</td>
<td>Parameters list for the page. This is a semicolon delimited String of name-value pairs. For example, &quot;param1=value1;param2=value2&quot;</td>
</tr>
<tr>
<td>taskFile</td>
<td>Y</td>
<td>Name of the task definition file. For example, /WEB-INF/task-flow-definition.xml. See Passing Parameters in Search.</td>
</tr>
<tr>
<td>taskName</td>
<td>Y</td>
<td>The task flow definition ID. Available from the task definition file &lt;task-flow-definition&gt; ID attribute. For example, &lt;task-flow-definition id='task-flow-definition'&gt; would be &quot;task-flow-definition&quot;. See Passing Parameters in Search.</td>
</tr>
<tr>
<td>navTaskKeyList</td>
<td>N</td>
<td>Key list to pass into the task flow to open in the target workspace. This is a semicolon delimited string of keys or key-value pairs. For example &quot;key1;key2=value2&quot;</td>
</tr>
<tr>
<td>navTaskParametersList</td>
<td>N</td>
<td>Parameters list to pass in to the task flow to open in the target workspace. This is a semicolon delimited string of name-value pairs. For example &quot;param1=value1;param2=value2&quot;</td>
</tr>
<tr>
<td>iconURL</td>
<td>N</td>
<td>URL of an icon to show next to the Action. Can be a relative reference such as /media/search/mime_doc.gif or a full URL such as <a href="http://host:port/path/to/icon.gif">http://host:port/path/to/icon.gif</a>.</td>
</tr>
<tr>
<td>toolTip</td>
<td>N</td>
<td>Tooltip of the action. This also is available for URL actions.</td>
</tr>
<tr>
<td>navTaskLabel</td>
<td>N</td>
<td>Label to show on the results tab. Set the tab title of the tab that is opened after clicking a search result action. If not set, it will use the Action Name (the value shown in the results).</td>
</tr>
<tr>
<td>webApp</td>
<td>Y</td>
<td>Attribute used to look up the host and port of the associated WorkArea or Dashboard from the Oracle Fusion Applications Functional Core (ASK) deployment tables. These tables are populated at deployment time through Oracle Fusion Functional Setup Manager tasks.</td>
</tr>
<tr>
<td>contextCode</td>
<td>N</td>
<td>Used for Recent Items striping of data by business line. This attribute will divide recent items into groups. Valid values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CommonDomain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• BI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• FIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SCM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CRM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• PRJ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• HCM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• PRC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• IC</td>
</tr>
<tr>
<td>objectType</td>
<td>N</td>
<td>Used for Recent Items striping of data by business line. This is a further subdivision within the contextCode classification for the recent item object (such as opportunity, contact, lead, quote). If objectType is supplied, contextCode should be set.</td>
</tr>
<tr>
<td>Name</td>
<td>Required</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>objectName</strong></td>
<td>N</td>
<td>User recognizable String for a given recent item instance. Used as string to match keystrokes entered in auto suggest field. This value will be in the language of the user at the point of navigation (that is, when the record is saved). It will not change if the user changes language. If <code>objectName</code> is supplied, <code>contextCode</code> and <code>objectType</code> also should be set.</td>
</tr>
</tbody>
</table>
| **FUSECardCode**  | N        | When navigating to FUSE, this is the `globalItemNodeId` of the global menu entry that describes the page to which to navigate. If this parameter is specified, `FUSEObjectCode` and `FUSETaskflowParamsList` also should be specified; these three parameters are passed to the `FndUIShellController.globalMenuNavigate()` API:  
    ```java
    public void globalMenuNavigate(String globalItemNodeId,  
                                   String taskItemNodeId,  
                                   String taskParametersList,  
                                   FndMethodParameters methodParameters)
    ```  
    This can be added as a parameter-value pair using the Figure 15-48 dialog. |
| **FUSEObjectCode**| N        | When navigating to FUSE, this is the `taskItemNodeId` of the global menu entry that describes the page to which to navigate. If this parameter is specified, `FUSECardCode` and `FUSETaskflowParamsList` also should be specified; these three parameters are passed to the `FndUIShellController.globalMenuNavigate()` method:  
    ```java
    public void globalMenuNavigate(String globalItemNodeId,  
                                   String taskItemNodeId,  
                                   String taskParametersList,  
                                   FndMethodParameters methodParameters)
    ```  
    This can be added as a parameter-value pair using the Figure 15-48 dialog. |
<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUSETaskflowParamsList</td>
<td>N</td>
<td>When navigating to FUSE, this is the FUSETaskflowParamsList of the global menu entry that describes the page to which to navigate. This parameter likely will contain a groovy expression that embeds the primary keys of the search result to allow navigation to the exact search result. If this parameter is specified, FUSECardCode and FUSEObjectCode also should be specified; these three parameters are passed to the FndUIShellController.globalMenuNavigate() API: public void globalMenuNavigate(String globalItemNodeId, String taskItemNodeId, String taskParametersList FndMethodParameters) This can be added as a parameter-value pair using the Figure 15-48 dialog.</td>
</tr>
<tr>
<td>FUSETaskFlowId</td>
<td>N (conditionally)</td>
<td>The taskId of the required dynamic navigation results flow in the form: /WEB-INF/oracle/apps/fnd/appcore/patterns/uishell/GlobalSearchDummyResultsTaskFlow.xml#GlobalSearchDummyResultsTaskFlow</td>
</tr>
<tr>
<td>FUSENavTaskKeyList</td>
<td>N</td>
<td>The task keylist used to identify dynamic tabs.</td>
</tr>
<tr>
<td>FUSENavTaskLabel</td>
<td>N</td>
<td>The label to show on dynamic tabs.</td>
</tr>
</tbody>
</table>

This information applies to the next three parameters: FUSETaskFlowId, FUSENavTaskKeyList and FUSENavTaskLabel.

Starting with Release 11, Applications Core ships a single FuseOverview page used by all Oracle Fusion applications. All content is supplied by the menus.

The pre-Release 11 navigation will continue to work, however there is a limitation on the taskflows that can be opened. That is addressed by these new parameters.

The navigation made by this call will only navigate to a defaultMain taskflow: FndUIShellController.globalMenuNavigate(FUSECardCode, FUSEObjectCode, FUSETaskflowParamsList, null)

This is okay for some teams because they open the results in a popup using adfc navigation flow cases built into their flows.

For teams coming new to FUSE search in Release 11, the requirement is to open a dynamicMain taskflow, much the same as in classic UIShell.

With the addition of these parameters a user can be sent to a dynamicMain taskflow.

There are now two use-cases for configuring FUSE Task Action parameters:

- The original defaultMain navigation:
Enter FUSECardCode, FUSEObjectCode, FUSETaskflowParamsList parameters only.

- New dynamicMain navigation:
  Enter FUSECardCode, FUSETaskflowParamsList, FUSETaskFlowId and, optionally, FUSENavTaskKeyList and FUSENavTaskLabel.
  Do not enter FUSEObjectCode.

The Classic UIShell and FUSE parameters can continue to co-exist in the same action.

---

**Caution:**
If you have a searchable view object with a task search action, the parameters passed to the task flow from FndUIShellController.navigate(...) will be Strings, not the native type of the view object attributes. You must ensure that these values are converted from their native type to a String (in the navTaskParametersList) and back correctly (in your task flow).

For Integer types, this is largely automatic (if you reference the parameter as a String in the task flow), but use caution for dates and decimals.

---

### 15.6.6.2.1 How to Determine the Navigation Target

Navigation depends on the defined parameters. You may have classic UIShell parameters, FUSE parameters (those starting with FUSE%) or both defined. The algorithm used to determine the navigation is:

```java
TaskSearchAction tsa = ...get the clicked action ...
boolean isClassicNav = get value of profile FND_CLASSIC_INTERFACE.
if isClassicNav
    // try to navigate to UIShell.
    if tsa has UIShellParameters
        UIShellNavigate(tsa);
    else if tsa hasFUSEParameters
        FUSEPlusNavigate(tsa);
else
    // log error, no nav can be done.
else
    // try to navigate to FUSE.
    if tsa has FUSEParameters
        FUSEPlusNavigate(tsa);
    else if tsa has UIShellParameters
        UIShellNavigate(tsa);
else
    // log error, no nav can be done.
```

### 15.6.6.2.2 How to Implement Preferred Navigation

Global Search will read the standard profile option FND_CLASSIC_INTERFACE that is used to determine where the user wants to be, and use the appropriate parameters to navigate. If there no parameters for that style of page, then it tries to use the other style, as shown in this example:

```java
TaskSearchAction tsa = ...get the clicked action ...
boolean isClassicNav = get value of profile FND_CLASSIC_INTERFACE.
if isClassicNav
    // try to navigate to UIShell.
    if tsa has UIShellParameters
        UIShellNavigate(tsa);
```
Global Search supports two parameters, `applicationStripe` and `pageDefinitionName`, in task search actions that, if they are present, change the definition of the other task action parameters used for navigation. These parameters all become “caret delimited.” That is, instead of having one value per parameter, they have multiple values, and they are delimited by the caret “^” character. In this case, all parameters must have the same number of delimited parts.

This additional configuration allows you as the developer to set up different navigation targets for the same action. The actual target followed will be determined when the user clicks the result and will be based on a permissions check based on the `applicationStripe` and `pageDefinitionName` parameters. If these two parameters are not supplied, the other parameters will be used “as is,” and navigation will be performed based on their values.

If the `applicationStripe` and `pageDefinitionName` parameter values are supplied, the algorithm used is the same as for tagging:

- Divide all delimited parameters based at the caret, and produce an ordered list of targets that can be opened.
- If there is only one target defined, use it with no permission check.
- For each target, determine if the user can open the page and task flow.
- If a target that can be opened is in the current view, use it.
- Take the first target that can be opened.

Whatever the outcome of the permissions check, you must ensure that at least one target can be opened, otherwise users will be presented with a blank page when they click the search result.

The parameters and descriptions for Preferred Navigation are shown in Table 15-13. Note that, although this table resembles Table 15-12, it presents a more complicated use case in which developers want to do a security check and direct the user to the most secure end point (the first allowed one in the list). The meaning of these columns changes with the caret delimitation; that is, a caret-delimited list of the old values, as well as two new parameters. Most users need to use only the information in Table 15-12.
### Table 15-13  Parameters for Preferred Navigation

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Delimited by caret</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationStripe</td>
<td>Y</td>
<td>N</td>
<td>(This attribute is used for pages.) Check security of the page against the policies that are located in LDAP. The applicationStripe name must be the same as the stripe name of the LDAP policy store, which is the same as the web.xml application.name attribute. If this parameter is supplied, the pageDefinitionName parameter must be supplied also. Example: crm^hcm</td>
</tr>
<tr>
<td>pageDefinitionName</td>
<td>N</td>
<td>Y</td>
<td>A delimited string of page definition names. If this parameter is supplied, the applicationStripe parameter also must be supplied. Example: oracle.apps.view.pageDefs.Test1PageDef^oracle.apps.view.pageDefs.AnotherPageDef</td>
</tr>
<tr>
<td>viewId</td>
<td>Y</td>
<td>Y</td>
<td>Name of the page for the pillar. This is shown in the browser URL bar. For example, in <a href="http://127.0.0.1:8989/context-root/faces/TestUIShellPage">http://127.0.0.1:8989/context-root/faces/TestUIShellPage</a>, it would be &quot;TestUIShellPage&amp;AnotherUIShellPage&quot;.</td>
</tr>
<tr>
<td>webApp</td>
<td>Y</td>
<td>Y</td>
<td>Attribute used to look up the host and port of the associated Work Area or Dashboard from the ASK deployment tables. These tables are populated at deployment time through Functional Setup Manager tasks.</td>
</tr>
<tr>
<td>pageParametersList</td>
<td>Y</td>
<td>N</td>
<td>Parameters list for the page. This is a semicolon delimited string of name-value pairs. For example, &quot;param1=value1;param2=value2^anotherParam1=value1;anotherParam2=value2&quot;</td>
</tr>
<tr>
<td>taskFile</td>
<td>Y</td>
<td>Y</td>
<td>Name of the task definition file. For example, &quot;/WEB-INF/task-flow-definition.xml^/WEB-INF/anotherTask-flow-definition.xml&quot;. See Passing Parameters in Search.</td>
</tr>
<tr>
<td>taskName</td>
<td>Y</td>
<td>Y</td>
<td>The task flow definition ID that is available from the task definition file &lt;task-flow-definition&gt; id attribute. For example, &lt;task-flow-definition id='task-flow-definition'&gt; would be &quot;task-flow-definition^another-task-flow-definition&quot;. See Passing Parameters in Search.</td>
</tr>
<tr>
<td>navTaskKeyList</td>
<td>N</td>
<td>Y</td>
<td>Key list to pass into the task flow to open in the target workspace. This is a semicolon delimited string of keys or key-value pairs. For example &quot;key1;key2=value2^anotherKey1;anotherKey2=value2&quot;</td>
</tr>
<tr>
<td>navTaskParametersList</td>
<td>N</td>
<td>Y</td>
<td>Parameters list to pass in to the task flow to open in the target workspace. This is a semicolon delimited string of name-value pairs. For example &quot;param1=value1;param2=value2^anotherParam1=value1;anotherParam2=value2&quot;</td>
</tr>
</tbody>
</table>
Table 15-13 (Cont.) Parameters for Preferred Navigation

<table>
<thead>
<tr>
<th>Name</th>
<th>Required</th>
<th>Delimited by caret &quot;^&quot;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>navTaskLabel</td>
<td>N</td>
<td>Y</td>
<td>The label to show on the results tab. (Set the tab title of the tab that is opened after clicking a search result action.) If not set, it will use the Action Name (the value shown in the results). For example: &quot;Manage user ^ View User&quot; (Note: This will be shown on the UI, so use resource bundles.)</td>
</tr>
<tr>
<td>iconURL</td>
<td>N</td>
<td>N</td>
<td>The URL of the icon to show next to the Action. It can be a relative reference, such as /media/search/mime_doc.gif or a full URL, such as <a href="http://host:port/path/to/icon.gif">http://host:port/path/to/icon.gif</a>.</td>
</tr>
<tr>
<td>Tooltip</td>
<td>N</td>
<td>N</td>
<td>A tooltip of action. This also is available for URL actions.</td>
</tr>
</tbody>
</table>

15.6.6.3 Passing Parameters in Oracle Fusion Applications Search

Ordinarily, taskFlowID uses the format <path><name>.xml#<name>; for instance taskFlowID="/WEB-INF/CaseDetails.xml#CaseDetails". However, Oracle Fusion Applications Search has taskFile and taskName attributes as shown in Figure 15-48. The code will merge them, adding the "," so they become <taskFile>#<taskName>.

Parameters always are passed as parameter name=value. Often, it is either a literal value or an expression to an attribute such as col1. Example 15-22 shows how to pass four parameters.

**Example 15-22 Parameter Passing in Oracle Fusion Applications Search**

```xml
<SearchResultActions>
  <Action
    Name="View Lookup Type"
    ActionType="Task"
    DefaultAction="true">
    <Title>
      <![CDATA["Lookup: " + Meaning]]>
    </Title>
    <ActionTarget>
      <![CDATA[null]]>
    </ActionTarget>
    <Parameters>
      <Parameter Name="navTaskParametersList">
        <Value>
          <![CDATA["lookupType=" + LookupType + ";viewApplicationId=" + ViewApplicationId + ";language=US;meaning=" + Meaning]]>
        </Value>
      </Parameter>
      <Parameter Name="webApp">
        <Value>
          <![CDATA["GlobalSearch"]]
        </Value>
      </Parameter>
      <Parameter Name="TaskFile">
        <Value>
          <![CDATA["/WEB-INF/LookupTypeSearchResultsTaskFlow.xml"]]
        </Value>
      </Parameter>
    </Parameters>
  </Action>
</SearchResultActions>
```
15.6.6.4 Ordering the Other Actions

In the ECSF search UI in JDeveloper, it is possible to define no action, or a single default action.

If a default action is defined, it will be used. The other actions will be shown in sorted order based on task title. If no default action is defined, the first sorted action will be used as the default action.

This sorting mechanism is used because there is no way, using the current ECSF APIs, to provide a stable order of actions.

Due to this sorting mechanism, it is strongly recommended to have stable, sortable task titles (they may be groovy bound and therefore mutate based on an individual search result) to prevent confusing the end user.

15.6.6.5 Using Click Path and the Saved Search

When the user is using Oracle Fusion Applications Search before saving a search, he or she may perform several interactions with the UI including:

- Expanding the attribute filters by selection (performs searches)
- Narrowing the search terms
- Opening unsearched groups (which performs searches in those groups)
- Scrolling through results in a group

This is called the click path of the user.

When a search is saved, some of this information (the structural part at the tip of the click path) is saved, but prior actions and exact scroll positions are not. This means that when running a saved search, the following items are not restored to the user:

- Exact expanded groups in the result at the time of save
- Scroll positions within a group
- Full LOV expansion state of attribute filters
When ECSF returns facet information, it returns facet entries only for the level below that which is selected. For example, if there are no filters, the facets will be shown correctly with one level of detail. If a first-level facet is selected, that selection will be shown, but not its siblings. If there are facets below that level, this next level will be shown as these are returned. As the user starts to refine or expand the attribute filters, the search filters will be filled in based on this new click path.

15.6.7 How to Integrate Non-Applications Data into Oracle Fusion Applications Search

Oracle Fusion Applications Search can also be used with non-standard ECSF searchable view objects.

15.6.7.1 Oracle Business Intelligence Integration

Oracle Business Intelligence results will be shown in a results area separate from Oracle Fusion Applications Search view object results. To implement this separation, Oracle Fusion Applications Search shows results in a multiple-tab format. The tabs are named Applications, where all Search view objects and WebCenter Portal results will reside, as well as Business Intelligence. The split is performed at a category (or searchable groups) level, so you will see a consolidated list of categories in the multi-select category dropdown. These categories are split at search time.

The business rule that splits categories into the Oracle Business Intelligence table is a lowercase category_name, such as `bi_%`.

Although the formatting of Oracle Business Intelligence results will be slightly different, no developer action is required.

**Oracle Secure Enterprise Search (SES) Setup**

To set up SES, you need to set up these parameters:

- **Source**
  
  See the Oracle Fusion Middleware Developer’s Guide for Oracle Business Intelligence Enterprise Edition guide for how to create your Oracle Business Intelligence source. Define a source based on the Oracle EBusiness Suite R12, and give the following parameters:

  SES Source Configuration:
  
  Configuration URL: `http://10.156.30.40:9704/bisearch/crawler/oracle.biee.search.BISearchableTreeObject/ConfigFeed?forceInitialCrawl=true`
  
  User ID: Administrator
  
  Password: password

  Authorization Tab:
  
  
  User ID: Administrator
  
  Password: password
  
  Business Component: `oracle.biee.search.BISearchableTreeObject`
  

  where the IP address is your Oracle Business Intelligence server installation, and the user name/password are for a sufficiently authorized Oracle Business Intelligence user.

  Leave all other values at default.

- **Source Group**
Create a source group (SES Searchtab then Source groups) and name it <some code name>.

You may go into Global Settings and translate the group name so the users see a more recognizable name.

Import the group as an external category into ECSF.

- **Searching**

  When you click a link, you will be redirected to Oracle Business Intelligence. If you do not have a consolidated Oracle Internet Directory (OID) setup, you will be asked to log in again.

**15.6.7.2 Ensuring Parity of Users**

Users must be defined in multiple applications if you do not have a single authentication store.

Ensure that you have a user defined that is common across both Fusion Applications and any external applications such as Oracle WebCenter Portal or Oracle Business Intelligence.

For instance, you can create an fmwadmin user on the Oracle Fusion Applications side by adding to the jazn-data.xml file.

With Oracle Secure Enterprise Search (Oracle SES) Authentication pointing to the ECSF SearchFeedServlet, which is using the WebLogic Server container security, this user will be verified by the Oracle SES authentication callbacks.

In a true enterprise environment, both the Oracle Fusion Applications web container and WebCenter Portal would be set up with the same OID.

Using two different authentication stores will mean you get multiple logins when clicking results.

**15.7 Implementing Auto-suggest**

The fnd:searchSuggestBehavior tag implements suggestion functionality, often known as typeahead. It is added to an af:inputText field, and will display as a popup menu below the field as a user types into the field. Its primary use is in Global Search to show groupings of suggestions from various sources, and for CRM to show suggestions in its search fields. The search suggest configuration for Global Search is done using the setup task flows described in How to Control the Look and Feel of Search. The tag is available in JDeveloper under the Applications Components in the Component Palette, as shown in Figure 15-49.
The suggestions in the group fall into two types of values:

- **Search suggestions.** These are keywords on which the user wants to search, and when selected, these words will be placed in the search field.

  **Note:** Keywords can be deleted using the Oracle Enterprise Scheduler Service submission UI. No additional coding is required.

- **Shortcuts to other functionality.** Selecting these entries will perform a specific function for that entry; the text will not be placed in the search field because it is not a keyword. For example, if the user types part of a Navigator entry, the Navigator link will be suggested to them; clicking it will navigate to that menu entry, just the same as if the user had selected the entry from the Navigator.

The `fnd:SearchSuggestBehavior` is developed as an Oracle ADF Behavior that closely follows the Oracle ADF version `af: autoSuggestBehavior`.

`fnd:searchSuggestBehavior` is similar in functionality to `af: autoSuggestBehavior` but adds:

- the rendering of group headings and items
- the expanded scrolling behavior to skip heading selection
- selection into the field only of keyword entries (not menu items)
- rendering of icons for each group and suggestion

It also adds to the communication to the server with new initialization, top suggestion and selection messages.

The user interacts with the suggestion list as follows:

- Auto-suggest starts with the third typed character. This can be configured.
- The user starts to type into the field against which the search suggest is defined.
- If there are no matching suggestions in a section, the section is not shown.
• As soon as the user selects the search field, a message is sent to the server to start to pre-load the lists of suggest items, so that by the time the user has typed the second character (expect a second or so), any delay on startup is done. As many groups as possible are loaded in a background thread to speed up the load of data.

• A user can navigate in the popup by using arrow keys, **Home** and **End** to circle among entries. As they do this, the currently-selected entry will be highlighted, and if it is a keyword, it will be entered in the search field. If it is a Recent Item or a Favorite, it will not be placed in the search field because it is not a search term.

• Pressing **Enter** will perform the search if the highlighted item is a search term. Otherwise, if it represents a navigation, the relevant task flow with parameters will be called. This can be configured.

• The match done is a *case insensitive* in string. The matched substring is highlighted in bold.

• A maximum of 25 items will be spread over the groups as evenly as possible. This can be configured.

• Any item that is to open a URL in a new window will show as a URL in the popup. This is required to set target="_blank" so the URL opens in a new browser window or tab, and not by overwriting the current Fusion Applications window.

• Groups are shown in the order defined in the JSPX tag.

Skinning keys are used for the items in the popup:

• .FndSearchSuggestItem

• .FndSearchSuggestItem:selected

• .FndSearchSuggestGroup

• .FndSearchSuggestGroup:first-child

• .FndSearchSuggestItemNormal

• .FndSearchSuggestItemCount

• .FndSearchSuggestItemCount:rtl

• .FndSearchSuggestItemHilight

The attributes of the `fnd:searchSuggestBehavior` tag are shown in Table 15-14.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>EL capable?</th>
<th>Required?</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Yes</td>
<td>No</td>
<td>Popup</td>
<td>The type of search suggest behavior. Valid values are inline and popup. Inline is where the suggestions will be placed in the content of an existing panel. This panel is identified by <code>inlineSuggestionComponentId</code>. The popup is the normal popup UI that will show below the search field. Popup is the default and the majority use case.</td>
</tr>
<tr>
<td>Attribute Name</td>
<td>EL capable?</td>
<td>Required?</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------</td>
<td>-----------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>maxSuggestedItems</td>
<td>Yes, resolves to an integer</td>
<td>No</td>
<td>25</td>
<td>The maximum suggested items to be shown across all groups. Items will be evenly distributed across groups to achieve this.</td>
</tr>
<tr>
<td>minCharCount</td>
<td>Yes, resolves to an integer</td>
<td>No</td>
<td>3</td>
<td>The minimum number of characters that need to be typed before the suggestions are shown.</td>
</tr>
<tr>
<td>suggestGroup&lt;x&gt; where x is a number between 1 and 8</td>
<td>Yes, resolves to a SearchSuggestGroup</td>
<td>Yes for suggestGroup1, No for suggestGroup2-8</td>
<td>N/A</td>
<td>A value that resolves to an oracle.apps.fnd.appcore.patterns.ui.behaviors.searchSuggest.SearchSuggestGroup, or a predefined group name. A group corresponds to the subheadings and items shown in the suggestion list, and are in the order of &lt;x&gt;. Predefined groups that can be referenced by code are: searchterms, globalsavedsearches, globalrecentsearches, navigator, recentitems, favoriteitems, watchlist. These predefined groups are only available to use within UI Shell pages.</td>
</tr>
<tr>
<td>ContextCode&lt;x&gt; where x is a number between 1 and 8</td>
<td>Yes</td>
<td>No</td>
<td>null</td>
<td>A context code for group x to provide meaning and subdivision of the group data. For Fusion Applications, these groups are expected. (For a custom group, this is group dependent.) • CommonDomain • BI • FIN • SCM • CRM • PRJ • HCM • PRC • IC</td>
</tr>
<tr>
<td>ObjectType&lt;x&gt; where x is a number between 1 and 8</td>
<td>Yes</td>
<td>No</td>
<td>null</td>
<td>A further division under contextCode&lt;x&gt; to provide further meaning within group x. For Fusion Applications, a meaningful object is expected, such as Contracts.</td>
</tr>
<tr>
<td>Attribute Name</td>
<td>EL capable?</td>
<td>Required?</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------</td>
<td>-----------</td>
<td>---------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>suggestGroups</td>
<td>Yes, resolves to</td>
<td>Yes</td>
<td>N/A</td>
<td>A list of Search Suggest Group Options. Each group option represents a group that will be shown in the autosuggest list, in the order created in the list.</td>
</tr>
<tr>
<td></td>
<td>public List&lt;SearchSuggestGroupOption&gt;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>typeDelay</td>
<td>Yes, resolves to</td>
<td>No</td>
<td>500</td>
<td>The delay between keystrokes before the suggestions are fetched. The number is a time in milliseconds.</td>
</tr>
<tr>
<td></td>
<td>an integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>suggestionSourceId</td>
<td>Yes, resolves to</td>
<td>No</td>
<td>null</td>
<td>A string that is unique to this source of suggestions. If two or more fields are to use the same source of suggestions, they can use the same identifier. In this case, they must all have the same value for the suggestedGroups attribute.</td>
</tr>
<tr>
<td></td>
<td>a string</td>
<td></td>
<td></td>
<td>suggestionSourceId will identify a session-level managed bean, so it must be unique in the range of session managed beans, and follow the rules of managed bean identifiers. If it is not set, it will use the clientId of the text component to which this behavior is added.</td>
</tr>
<tr>
<td>submitOnEnterComponentId</td>
<td>Yes, resolves to</td>
<td>No</td>
<td>null</td>
<td>A relative Component ID for an action Component that is to be submitted when the user presses Enter in this field. This provides the equivalent of pressing this button. If this is not set, pressing Enter will have no effect in the field, unless other listeners process this keypress.</td>
</tr>
<tr>
<td></td>
<td>a string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inlineSuggestionComponentId</td>
<td>Yes, resolves to</td>
<td>No</td>
<td>Null</td>
<td>A relative Component ID for the suggestions to be placed instead of in a popup under the field. This attribute is required if type=inline. This component needs to have clientComponent=true set and should be visible=false.</td>
</tr>
<tr>
<td></td>
<td>a string</td>
<td></td>
<td></td>
<td>This component needs to be a container component; panelGroupLayout is recommended.</td>
</tr>
<tr>
<td>inlineReplaceComponentId</td>
<td>Yes, resolves to</td>
<td>No</td>
<td>null</td>
<td>A relative Component ID representing a component to hide if type=inline. If set, this is a way to swap out the UI and replace it with that shown in inlineSuggestionComponentId.</td>
</tr>
<tr>
<td></td>
<td>a string</td>
<td></td>
<td></td>
<td>This component needs to have clientComponent=true set.</td>
</tr>
<tr>
<td>Attribute Name</td>
<td>EL capable?</td>
<td>Required?</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>-----------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>topSuggestions</td>
<td>Yes, resolves to a string</td>
<td>No</td>
<td>On</td>
<td>Determines whether to show top suggestions when the field is first clicked. These suggestions are provided without filters and provide a starting point for the user. Once the user starts to type, regular filtered selections are shown. This is only processed if type=popup, as inline suggestions handle this using a dedicated panel identified by inlineSuggestionComponentId. Valid values are on and off. Off is the default.</td>
</tr>
<tr>
<td>postSelectionFocusComponentId</td>
<td>Yes, resolves to a string</td>
<td>No</td>
<td>null</td>
<td>A relative Component ID representing a component to give focus after a selection is made. This is primarily of use so that focus is no longer in the search field, and topSelections will be shown immediately when the user re-enters the field. By default, focus will remain in the search field.</td>
</tr>
<tr>
<td>adfPopupId</td>
<td>No</td>
<td></td>
<td></td>
<td>The relative component Id of an Oracle ADF popup in which inline suggestions are to be shown. This popup must have contentDelivery=&quot;immediate&quot; and childCreation=&quot;immediate&quot; so that it is immediately available on the client. The Oracle ADF popup should have a panel with Id corresponding to inlineSuggestionComponentId and may have inlineReplaceComponentId and adfMessageComponentId components. This attribute is only relevant when type=&quot;inline&quot;.</td>
</tr>
<tr>
<td>adfMessageComponentId</td>
<td>No</td>
<td></td>
<td></td>
<td>A relative Component Id of a component on which to set a &quot;search for 'linda'&quot; message. This attribute is only relevant when type=&quot;inline&quot;.</td>
</tr>
<tr>
<td>adfShowComponentIds</td>
<td>No</td>
<td></td>
<td></td>
<td>A space-separated list of relative Component Ids representing components to make visible once the loading suggestions message is finished. This attribute is only relevant when type=&quot;inline&quot; and adfPopupId is set, and is a way of not allowing the user to select options on the inline popup that are not yet ready to be selected because their data is not initialized (for example clicking a button).</td>
</tr>
<tr>
<td>noMatchMessage</td>
<td>No</td>
<td></td>
<td></td>
<td>The message shown to the user when there are no matches. If this attribute is not set, no message will be shown, and any existing suggestions will be hidden.</td>
</tr>
</tbody>
</table>
### Table 15-14 (Cont.) find:searchSuggestBehavior Attributes

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>EL capable?</th>
<th>Required?</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupIcons</td>
<td>No</td>
<td></td>
<td></td>
<td>Do you want to show the icons in group headings for all defined groups? This is an override for the group-specific generation of icons in group headings. groupHeadings must be 'on' (the default) for this attribute to have any effect, because the icons are shown in the heading. If a group does not have a heading icon, this setting will have no effect because there is nothing to show or hide. However if the group does have a heading icon, this setting will show or hide it. Valid values are 'on' and 'off'.</td>
</tr>
<tr>
<td>adfPopupId</td>
<td>No</td>
<td></td>
<td></td>
<td>This attribute is only relevant when type=&quot;inline&quot;. The relative component Id of an ADF popup in which inline suggestions are to be shown. This popup must have contentDelivery=&quot;immediate&quot; and childCreation=&quot;immediate&quot; so that it is immediately available on the client. The ADF popup should have a panel with Id corresponding to inlineSuggestionComponentId and may have inlineReplaceComponentId and adfMessageComponentId components.</td>
</tr>
<tr>
<td>adfMessageComponentId</td>
<td>No</td>
<td></td>
<td></td>
<td>This attribute is only relevant when type=&quot;inline&quot;. A relative component Id of a component on which to set a &quot;search for search_term&quot; message.</td>
</tr>
<tr>
<td>adfShowComponentIds</td>
<td>No</td>
<td></td>
<td></td>
<td>A space-separated list of relative component Ids representing components to make visible once the loading suggestions message is finished. This attribute is only relevant when type=&quot;inline&quot; and adfPopupId is set, and is a way of not allowing the user to select options on the inline popup that are not yet ready to be selected because their data is not initialized (for example clicking a button).</td>
</tr>
<tr>
<td>noMatchMessage</td>
<td>No</td>
<td></td>
<td></td>
<td>The message shown to the user when there are no matches. If it is not set, no message will be shown and any existing suggestions will be hidden.</td>
</tr>
</tbody>
</table>

These will be used by Global Search in the 'Global Search Field with Results Popup' taskflow to show suggestions and to provide personalization UI (by clicking the gear icon).

Additional behaviors are:

- Top Suggestions Loading. If topSuggestions='on' and this is the first time the search field has been clicked, then static loading UI will be shown.

- Show More. If there is more than one group of suggestions and if any of those groups have been truncated to fit within the maximum number of suggestions, a Show More link will be shown on each group. Once clicked, only this group's selections will be shown with a Show Less link.
• If no suggestions are found, the noMatchMessage is shown. If this is empty or
null, the suggest popup will not be shown. If it is already open, it will be closed. If
type='inline', the suggest content will be hidden; if
inlineReplaceComponentId is specified, this content will be re-showed.

• If adfMessageComponentId is specified, a message "Search for 'contents of
search field'" will be set as the text of that component (on the client).

• If adfShowComponentIds are specified, these will be made visible after the top
suggestions loading banner has been shown.

• If in screen reader mode, fnd:searchSuggestBehavior will no longer be
disabled if type='inline' (it is still disabled if type = 'popup').

• Top suggestions will be shown all the time using the defined suggest groups. This
means the suggest content is completely consistent and symmetrical for top and
autosuggest content in both types of popup and inline.

15.7.1 How to Implement fnd:searchSuggestBehavior

To implement fnd:searchSuggestBehavior, the basic developer steps are:

• Drag-and-drop the component onto an af:inputText field.

• Set the appropriate attributes. suggestGroups is required.

• If creating a custom group, extend AbstractSearchSuggestGroup, and
reference it using Expression Language.

These examples show how you can implement fnd:searchSuggestBehavior.

Using suggestGroups containing a predefined and custom group

```xml
<af:inputText id="SearchField"
value="#{pageFlowScope.searchString}"
maximumLength="2048"
required="false" simple="true" autoComplete="off"
clientComponent="true" usage="search">
    <fnd:searchSuggestBehavior suggestGroups="#{SomeBean.searchSuggestBehaviorGroups}"/>
</af:inputText>

/**
 * Get the Search Behavior groups.
 * This is based on the GlobalSearchOptions set for the current page. If no
 * options are set it defaults based on the method
 * GlobalSearchOptions.getDefaultGlobalSearchOptions().
 * @return List of behavior groups.
 */
public List<SearchSuggestGroupOption> getSearchSuggestBehaviorGroups()
{
    List<SearchSuggestGroupOption> ret = new ArrayList<SearchSuggestGroupOption>();
    ret.add(new SearchSuggestGroupOption(KeywordSearchSuggestGroup.NAME));
    ret.add(new SearchSuggestGroupOption("#{SomeBean.customGroup}",
            GroupHeading.ON,
            "/path/to/icon.png",
            "Group Heading",
            null,
            null));

    return ret;
}
```
Implementing Auto-suggest

public AbstractSearchSuggestGroup getCustomGroup()
{
    // This group will have any heading name, icon,
    // context code and object code set on it by the framework
    // to override code (developer set) values.
    return new MyCustomSearchSuggestGroup();
}

Simple popup with all predefined groups

Simple Default: 

Popup with 3-second delay

Popup with maximum of 10 suggestions

Popup with submit on Enter

Oracle Fusion Applications Developer's Guide
Implementing Auto-suggest

Common data for two suggestion popups

```xml
<af:inputText id="it6" label="common behavior 1">
    <fnd:searchSuggestBehavior suggestionSourceId="common1"
        suggestGroup1="searchterms"
        suggestGroup2="navigator"/>
</af:inputText>

<af:inputText id="it7" label="common behavior 2">
    <fnd:searchSuggestBehavior suggestionSourceId="common1"
        suggestGroup1="searchterms"
        suggestGroup2="navigator"/>
</af:inputText>
```

Custom group with topSuggestions

```xml
<af:inputText id="topSuggestions" label="Top Suggestions: " value="seahawk">
    <fnd:searchSuggestBehavior
        suggestGroup1="#{SuggestGroupFactory.bookSearchSuggestGroup}"
        topSuggestions="on"/>
</af:inputText>
```

Where SuggestGroupFactory is some Expression Language-reachable managed bean:

```java
public class SuggestGroupFactory
    implements Serializable
{
    public SearchSuggestGroup getBookSearchSuggestGroup()
    {
        return new BookSearchSuggestGroup();
    }
}
```

Group that uses object striping

```xml
<af:inputText id="crmRIMenu1" label="CRM RI Menu" value="Lin">
    <fnd:searchSuggestBehavior
        suggestGroup1="recentitems"
        contextCode1="CRM"
        objectType1="Search"/>
</af:inputText>
```

Group that hides and shows content in the base page

The following suggestion will be shown in the "suggestions" panelgrouplayout. The content in "swapout" will be hidden when suggestions are shown.

```xml
<af:inputText id="suggestinline" label="Suggest inline:">
    <fnd:searchSuggestBehavior
        inlineSuggestionComponentId="suggestions"
        type="inline"
        inlineReplaceComponentId="swapout"
        suggestGroup1="#{SuggestGroupFactory.bookSearchSuggestGroup}"
        contextCode1="Book"
        objectType1="theseahawk.txt"
        noMatchMessage="No suggestions found."
        minCharCount="3"/>
</af:inputText>
```

```xml
<af:panelGroupLayout layout="vertical" id="suggestions" rendered="true"
    styleClass="fndGlobalSearchSuggestPanel"
    clientComponent="true" visible="false">
<!-- Autosuggest content will be added here. -->
</af:panelGroupLayout>
```
15.7.2 How to Implement a Custom Suggestion Group

If you wish to create your own list of suggestions to show the user, you need to create a SearchSuggestGroup and register it with the af:searchSuggestBehavior tag for the af:inputText field that will show the suggestions.

Before you begin, you must implement the oracle.apps.fnd.applcore.patterns.ui.behaviors.searchSuggest.SearchSuggestGroup interface by extending the AbstractSearchSuggestGroup interface directly; this is not supported.

This interface is available in the UICOMPONENTS-View.jar, which is available by including the Applications Core (ViewController) library in your project. This dependency should be met already.

The developed class, and all resources it needs, should be available to all applications that consume the functionality. For example, for CRM implementation, it should be available in common CRM libraries. URL references used for icons in the group headings, or for suggestion items, should not be local to the web application unless the icon comes from a common icon repository that is available across web applications.

BookSearchSuggestGroup

This is an example implementation that loads a book from a text file, word counts it and provides the words as suggestions.

```java
public class BookSearchSuggestGroup extends AbstractSearchSuggestGroup {
    /**
     * Unique name of this group. Must be unique across all groups.
     */
    public static final String NAME = "booksearchterms";
    private List<SearchSuggestItem> _keywords = new ArrayList<SearchSuggestItem>();

    public BookSearchSuggestGroup() {
        super();
    }

    @Override
    public String getDisplayName() {
        return "My Book Terms";
    }

    @Override
    public String getIconURL() {
        return "images/applcore/uishell/globalSearch/search_ena.png";
    }

    @Override
    public List<SearchSuggestItem> getItems(String searchText,
```
int maxSuggestedItems)
{
    long now = System.currentTimeMillis();
    List<SearchSuggestItem> ret = new ArrayList<SearchSuggestItem>();

    String upperFilter = searchText.toUpperCase();
    int size = _keywords == null ? 0 : _keywords.size();
    for (int i = 0; i < size; i++)
    {
        SearchSuggestItem kw = _keywords.get(i);
        String keyword = kw.getText();
        String upperKw = keyword.toUpperCase();
        int index = upperKw.indexOf(upperFilter);
        if (index >= 0)
        {
            ret.add(new SearchSuggestItem(keyword, kw.getCount()));
            if (ret.size() >= maxSuggestedItems)
                break;
        }
    }

    if (AppsLogger.isEnabled(AppsLogger.FINEST))
    {
        long later = System.currentTimeMillis();
        long duration = later - now;
        AppsLogger.write(this,
            "Global Search: BookSearchSuggestGroup, getItems: Found: " +
            ret.size() + " items for: " + searchText + " in " + duration +
            " mSecs.",
            AppsLogger.FINEST);
    }
    return ret;
}

@Override
public List<SearchSuggestItem> getTopSuggestions(String searchText,
    int maxSuggestedItems)
{
    long now = System.currentTimeMillis();
    List<SearchSuggestItem> ret = new ArrayList<SearchSuggestItem>();

    ret.add(new SearchSuggestItem("sea"));
    ret.add(new SearchSuggestItem("seahawk"));
    ret.add(new SearchSuggestItem("Rafael"));
    ret.add(new SearchSuggestItem("Sabatini"));
    ret.add(new SearchSuggestItem("pirate"));
    ret.add(new SearchSuggestItem("English"));
    ret.add(new SearchSuggestItem("Spanish"));
    ret.add(new SearchSuggestItem("privateer"));
    ret.add(new SearchSuggestItem("Armada"));

    Collections.sort(ret);

    if (AppsLogger.isEnabled(AppsLogger.FINEST))
    {
        long later = System.currentTimeMillis();
        long duration = later - now;
        AppsLogger.write(this,
            "Global Search: BookSearchSuggestGroup, getTopSuggestions:
            Implementing Auto-suggest
            Implement Search Functions in the UI Shell
            15-111"};
Implementing Auto-suggest

15.7.3 How to Implement fnd:searchSuggestBehavior in Global Search

The Search Suggest functionality has been implemented in Applications Core Global Search; there is nothing the developer needs to do to get this feature.

By default, the user will see predefined sources of suggestions as described in each suggestion group described in Table 15-15.
Table 15-15  Search Suggestions Sections

<table>
<thead>
<tr>
<th>Section</th>
<th>Live data</th>
<th>Supports Striping using contextCode, objectType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search Terms</td>
<td>Essentially Yes. Data is loaded at start but updated in memory. Data is saved to table on each Search.</td>
<td>No</td>
</tr>
<tr>
<td>Recent Items</td>
<td>Yes. Data is the same as that in the Recent Items list. Data is updated live in the list and saved on logout.</td>
<td>Yes</td>
</tr>
<tr>
<td>Favorite Items</td>
<td>No, loaded up front.</td>
<td>No</td>
</tr>
<tr>
<td>Navigator</td>
<td>No, loaded up front.</td>
<td>Yes</td>
</tr>
<tr>
<td>Watchlist</td>
<td>No, loaded up front.</td>
<td>No</td>
</tr>
<tr>
<td>Saved Global Searches</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Recent Global Searches</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

My Search Terms

Alternate search words that will be placed in the search field if the user clicks them.

The user then can perform a search by pressing Enter or clicking the Search icon.

The keywords are those from previous unfiltered searches by the user, using the result count of that search. These values are stores in a new table:

CREATE TABLE FND_FUSION_SEARCH_KEYWORDS
(
    FUSION_SEARCH_KEYWORDS_ID NUMBER(18,0) NOT NULL,
    USER_ID VARCHAR2(64 CHAR) NOT NULL,
    KEYWORDS VARCHAR2(260 CHAR) NOT NULL,
    SEARCH_COUNT NUMBER(18,0) NOT NULL,
    RESULT_COUNT NUMBER(18,0),
    ENTERPRISE_ID NUMBER(18,0) DEFAULT NVL(SYS_CONTEXT('FND_VPD_CTX', 'ENTERPRISE_ID'), 1) NOT NULL,
    CREATION_DATE TIMESTAMP (6) NOT NULL,
    CREATED_BY VARCHAR2(64 CHAR) NOT NULL,
    LAST_UPDATE_DATE TIMESTAMP (6) NOT NULL,
    LAST_UPDATED_BY VARCHAR2(64 CHAR) NOT NULL,
    LAST_UPDATE_LOGIN VARCHAR2(32 CHAR),
    CONSTRAINT FND_FUSION_SEARCH_KEYWORDS_PK PRIMARY KEY (FUSION_SEARCH_KEYWORDS_ID, ENTERPRISE_ID)
) DISABLE

My Recent Items

Recent Items from the recent items menu that match the search String.

When the user clicks a recent item, the task flow opens up directly.

The value shown to the user is the objectName, if present, otherwise it is the taskflow label (what is shown in the Recent Items UI). Striping is possible using contextCode and objectType if present. Note that Global Search does not use the striping values.
My Favorite Items
This is similar to recent items, however the favorite items need to be loaded from tables when first used and are thereafter cached in memory.

My Navigator
The Navigator items are loaded by using the same APIs the navigator menu uses, and will reuse any loaded or cached menu structure.

The value shown is the itemNode label. Striping is possible using contextCode and objectType if present. Note that Global Search does not use the striping values.

Support entries that are navigation targets (the FunUIShellController.navigate(...)), as well as pure URLs (<a href=...>) that open in a new frame. URLs will be shown in the popup as URLs; that is, with the underline.

My Watchlist
Watchlist items are loaded when first referenced.

Clicking an item expands into the watchlist item.

Saved Searches
Saved searches are referenced directly from Global Search.

Clicking an item runs the search.

Recent Searches
Recent searches are referenced directly from Global Search.

Clicking an item runs the search.

15.7.4 How to Implement Dynamic UI Support for searchSuggestBehavior

Dynamic UI support for searchSuggestBehavior satisfies these three requirements:

1. Dynamic configuration adds Java level configuration for fnd:searchSuggestBehavior. This allows developers to add input text fields dynamically through code using metadata available in MDS.

2. Enables table support for fnd:searchSuggestBehavior so, for example, developers can add flexfields to tables. fnd:searchSuggestBehavior works in repeating and stamped structures, and with various combinations of rowSelection.

3. Can show one set of suggestions but return a different value to the field. To provide alias lookup support, suggestions will be shown only when a prefix character (# by default) is typed. The suggestions shown are aliases and may show different values; they will not show the prefix character. When a suggestion is picked, the prefixed value is returned to the field.

User Interaction
User interaction for requirements 1 and 2 are the same as the existing fnd:searchSuggestBehavior solution. That is, the user clicks in an af:inputText field, the popup shows, and the user selects from it.

For requirement 3, the interaction is:
• User selects the field and types the prefix character ('#', though it could be anything). If the user does not type the prefix character, no suggestions are shown.

• User types another character and autosuggest shows suggestions based on the second character. The '#' is not part of the suggestions, which may not even contain the characters typed.

• As the user continues to type, the suggestions narrow.

• User selects a value and an alias value for the selection is returned to the field. This value will contain the prefix character and may not look like the suggestion selected.

Requirement 1 - Dynamic Configuration Design
Provides a factory and a bean for setting all suggest attributes, and the ability to apply to an af:inputText field.

class: SearchSuggestBehaviorFactory
method: createSearchSuggestBehaviorBean()

Class SearchSuggestBehaviorBean is obtained from SearchSuggestBehaviorFactory.

methods:
• applyTo(RichInputText it)
• setXXXX(ValueExpression ve) (1 method per attribute you can see in the property inspector for fnd:searchSuggestBehavior).

Use as shown in this example.

RichInputText it = .. somehow get an input text field.
SearchSuggestBehaviorBean ssb = SearchSuggestBehaviorFactory.createSearchSuggestBehaviorBean();
ssb.setSuggestGroup1(ssb.createValueExpression("oracle.apps.fnd.searchSuggest.ReflectiveSearchSuggestGroup"));
ssb.setContextCode1(ssb.createValueExpression("ContextCode" + numChildren));
ssb.setObjectTyep1(ssb.createValueExpression("ObjectCode" + numChildren));
ssb.setTopSuggestions(ssb.createValueExpression("on"));
... set other attributes as appropriate ...
ssb.applyTo(it);

Requirement 2 - Enable Table Support
The developer can get the clientId of the input text field on which the search suggest field is configured.

For tables and other repeating structures, the af:inputText field and fnd:searchSuggestBehavior are defined once in a column (for an af:table) and only exist once in the server-side JSF component tree. When the markup is rendered, it is stamped out repeatedly for the number of rows in question. Numeric Ids are added into the client ID of the component (:0 for row 0, :1 for row 1, and so on).

The following APIs are added to AbstractSearchSuggestGroup (the class all suggestion group writers must extend):

/**
 * Set the suggestFieldClientId. This is called by the search suggest framework
* when suggestions are requested for a field, and before the groups are polled for * those suggestions. This value can be used to distinguish between multiple fields * sharing the same suggestion data, and perhaps different columns in a table. * @param suggestFieldClientId client id, for example pt1:USma:0:MAt2:1:pt1:t1:1:it1 */

public void setSuggestFieldClientId(String suggestFieldClientId)
{
    _suggestFieldClientId = suggestFieldClientId;
}

/**
 * Get the suggestFieldClientId. This will be set by the default init() method,
 * and updated on every click in the suggest field.
 * @return suggest field client id of the form a:b:c.
 */
public String getSuggestFieldClientId()
{
    return _suggestFieldClientId;
}

The suggestion group developer then can use logic similar to that below to determine the row. If the developer is using a table with rowSelection="single", then common rowSelection APIs can be used (id1 below). However if any other rowSelection method is used, this will be unreliable. A variant of Method 2, below, can be used.

/**
 * Use the clientId to find the InputText component, the enclosing table and
 * thereby the selected row.
 * @return selected row, -1 if cannot find for any reason.
 */
private int getSelectedRow()
{
    // There are 2 ways to find the cell: one is to know which column we are
    // in by being told (I use contextCode here) and determining the row using
    // the table's selected row. However this requires the table to have
    // rowSelection="single" which is likely but not always the case.
    // The second is to use the clientId of the component and a bit of internal
    // knowledge of the encoding of the ids.

    int id1 = getSelectedRowUsingTable();
    int id2 = getSelectedRowUsingClientId();
    System.out.println("TableCellSuggestGroup: getSelectedRow: found row using table
    selected row: " + id1 + 
        ", by clientId: " + id2);
    if (id1 != id2)
    {
        if (AppsLogger.isEnabled(AppsLogger.WARNING))
        {
            AppsLogger.write(this,
                "TableCellSuggestGroup: getSelectedRow: found differing rows,
            row using table selected row: " + id1 + 
                ", by clientId: " + id2,
                AppsLogger.FINEST);
        }
        // client id is more accurate across different rowSelection attribute settings.
        return id2;
    }

    // client id is more accurate across different rowSelection attribute settings.
    return id2;
}

private int getSelectedRowUsingTable()
{ // Method 1:
UIComponent field = SearchUtils.findComponentByClientId(_searchFieldClientId);

if (field != null)
{
RichTable table =
(RichTable)PatternsUtil.getParentByType(field, RichTable.class);

if (table != null)
{
    // find the selected row.
    RowKeySet selectedRowKeys = table.getSelectedRowKeys();
    if (selectedRowKeys != null && selectedRowKeys.size() > 0)
    {
        // This works for Java ArrayLists, but will not work for BC collections.
        Integer row = (Integer)selectedRowKeys.iterator().next();
        return row;
    }
}
return -1;
}

private int getSelectedRowUsingClientId()
{
    // Method 2:
    String clientId = getSuggestFieldClientId();
    String[] clientIds = clientId.split("::");
    if (clientIds.length > 2)
    {
        return Integer.valueOf(clientIds[clientIds.length - 2]);
    }
    return -1;
}

Requirement 3 - Alias Lookup Design
This requirement is met by the suggestion group developer, with one enhancement to allow the transposition of the display and Id values.

find:searchSuggestBehavior supports the ability to have a displayed value with a different id behind the scenes. It is achieved by the suggestion group developer adding suggestions using one of the more detailed constructors of SearchSuggestItem, using a different Id and text:

/**
 * Create an Item.
 * @param text translatable text shown to user.
 * @param id unique identifier.
 * @param count Optional count.
 * @param url fully qualified url the user can click. Must be of the form http://
 * it will result in a html href.
 */

public SearchSuggestItem(String text, String id, Number count, String url)

Until this point, the Id could be used only when the AbstractSuggestGroup suggestions were submitted to the server as a shortcut, such as selecting a navigator menu item.
In this case the selected suggestion was not placed in the search field (this is achieved by over-riding `requiresSubmit()` and returning true).

If different Ids and text were returned as suggestions and `requiresSubmit()` returned false, the text would be shown and placed into the search field when selected. This is the typical search terms use case commonly seen with auto suggest.

A new method added to the `AbstractSuggestGroup` indicates that the value to be returned to the field is the Id and not the text. This can be used only when `requiresSubmit()` is false:

```java
/**
 * Given `placeSelectedItemIdInField()` returns true, do we place the SearchSuggestItem Id in the search field, or in the display value.
 * It is almost always the case that the display value will be used. However it is possible to perform a transposition where the user sees one thing in the list of the suggestions, but another is returned to the field.
 * An example of this is for typing aliases. The user may type a prefix character into the field to trigger suggest, such as '#'. The suggest group responds only when # is entered and it will then suggest a list of descriptive values without the '#' (they may be completely different to what is typed).
 * When the value is selected by the user, the #value is placed in the field, not the value seen in the suggest popup.
 * If this is set to true and no Id is set, then the display value is used as normal.
 * This does not apply if `requiresSubmit()` is true. In this case, if it is available, the Id is always sent.
 * @return true if to place the id and not the display value in the field when selected.
 * @see #placeSelectedItemIdInField
 * @see #requiresSubmit
 * @see SearchSuggestItem
 */
public boolean placeSelectedItemIdInField()
{
    return false;
}
```

The suggestion group developer has complete control of the rest of the requirements for this use case:

- In the `getItems(String searchText, int maxSuggestedItems)` method, only return suggestions when the `searchText` starts with the prefix character.
- Configure the `fnd:SearchSuggestBehavior` to start the suggestion at character 2 (the `minCharCount` attribute).
- Return `SearchSuggestItems` with a different Id and text. The text will be shown in the suggestion list.
- Override `placeSelectedItemIdInField` and return true to put the Id, not the text, into the field when selected.

**Example Dynamic Suggestion Group in a Form Field**

```java
public void addTextField(ActionEvent actionEvent)
{
    FacesContext context = FacesContext.getCurrentInstance();
    Application application = context.getApplication();
```
UIComponent btn = actionEvent.getComponent();
UITComponent pfl = PatternsUtil.getParentByTypeAndId(btn, RichPanelFormLayout.class, "suggestPanel");

int numChildren = pfl.getChildren().size();
RichInputText it = (RichInputText)
application.createComponent(RichInputText.COMPONENT_TYPE);
it.setId("it" + numChildren);
ValueExpression value = createValueExpression("#{pageFlowScope.DynamicSuggestInFormBean.it" + numChildren + "}");
it.setValueExpression("value", value);
it.setLabel("Input Text " + numChildren + ":");
it.setImmediate(true);
MethodBinding vcl = createMethodBinding("#{pageFlowScope.DynamicSuggestInFormBean.valueChanged}");
it.setValueChangeListener(vcl);

SearchSuggestBehaviorBean ssb = SearchSuggestBehaviorFactory.createSearchSuggestBehaviorBean();
ssb.setSuggestGroup1(ssb.createValueExpression("oracle.apps.fnd.searchSuggest.Prefixe
dSearchSuggestGroup");
ssb.setContextCode1(ssb.createValueExpression("ContextCode" + numChildren));
ssb.setObjectType1(ssb.createValueExpression("ObjectType" + numChildren));
ssb.setTopSuggestions(ssb.createValueExpression("on"));
ssb.setMinCharCount(new ConstantExpression(2));
ssb.applyTo(it);
pfl.getChildren().add(it);

Example Dynamic Suggestion Group in a Table Field

public void generateContent(ActionEvent e)
{
    _contentGenerated = true;
    for (int i = 0; i < _numTextFields; i++)
    {
        FacesContext context = FacesContext.getCurrentInstance();
        Application application = context.getApplication();
        UIViewRoot uiRoot = context.getViewRoot();

        RichColumn col = (RichColumn)PatternsUtil.findComponentByTypeAndId(uiRoot, RichColumn.class, "c" + i);
        RichInputText it = (RichInputText)
        application.createComponent(RichInputText.COMPONENT_TYPE);
        it.setId("it" + i);
        ValueExpression value = createValueExpression("#{row.it" + i + "}");
        it.setValueExpression("value", value);

        SearchSuggestBehaviorBean ssb = SearchSuggestBehaviorFactory.createSearchSuggestBehaviorBean();
        ssb.setSuggestGroup1(ssb.createValueExpression("oracle.apps.fnd.searchSuggest.TableCe
llSuggestGroup");
        ssb.setContextCode1(ssb.createValueExpression("" + i));
        ssb.setObjectType1(ssb.createValueExpression("ObjectCode" + i));
        ssb.setTopSuggestions(ssb.createValueExpression("on"));
        ssb.applyTo(it);
Implementing Auto-suggest

col.getChildren().add(it);
}
}

Example of a Table Cell Search Suggestion Group
package oracle.apps.fnd.searchSuggest;
import
import
import
import
import

java.util.ArrayList;
java.util.List;
java.util.Map;
javax.faces.component.UIComponent;
oracle.adf.view.rich.component.rich.data.RichTable;

import oracle.apps.fnd.applcore.globalSearch.ui.SearchUtils;
import oracle.apps.fnd.applcore.log.AppsLogger;
import
oracle.apps.fnd.applcore.patterns.ui.behaviors.searchSuggest.AbstractSearchSuggestGro
up;
import
oracle.apps.fnd.applcore.patterns.ui.behaviors.searchSuggest.SearchSuggestItem;
import oracle.apps.fnd.applcore.patterns.ui.util.PatternsUtil;
import org.apache.myfaces.trinidad.model.RowKeySet;
/**
* A suggest group for placing in a table cell.
* It uses the contextCode as the column number, and will determne the row from
* the selected row of the current table.
*/
public class TableCellSuggestGroup
extends AbstractSearchSuggestGroup
{
/**
* Unique name of this group. Must be unique across all groups.
*/
public static final String NAME = "TableCellSuggestGroup";
private static final String _NUMBERS =
"012345678901234567890123456789012345678901234567890123456789012345678901234567890123
4567890123456789";
private String _searchFieldClientId = null;
public TableCellSuggestGroup()
{
super();
}
/**
* {@inheritDoc}
*/
@Override
public String getDisplayName()
{
String overRiddenDisplayName = super.getDisplayName();
if (overRiddenDisplayName != null)
{
return overRiddenDisplayName;
}
return "Table Cell Suggestions";

15-120 Oracle Fusion Applications Developer's Guide


@Override
public String getName()
{
    String overRiddenName = super.getName();
    if (overRiddenName != null)
    {
        return overRiddenName;
    }

    return NAME;
}

@override
public void init(Map<String, Object> attributes)
{
    super.init(attributes);
    // Grab the client id, used for determining the row.
    _searchFieldClientId = (String) attributes.get("clientId");
}

/*
 * Get suggestions from user typing. In this case, it will return the row and
column
 * of the cell clicked, then the search text itself, and then a number
 * of suggestions based on the searchText appended with numbers.
 * @param searchText user typed text.
 * @param maxSuggestedItems maximum suggestions to bother returning, any more will
 * be ignored.
 * @return Suggestions to show the user.
 */
public List<SearchSuggestItem> getItems(String searchText,
                                        int maxSuggestedItems)
{
    List<SearchSuggestItem> ret = new ArrayList<SearchSuggestItem>();
    ret.add(new SearchSuggestItem(getCell()));
    ret.add(new SearchSuggestItem(searchText));
    for (int i = 1; i < (maxSuggestedItems - 1); i++)
    {
        ret.add(new SearchSuggestItem(searchText + " " + _NUMBERS.substring(0, i)));
    }

    return ret;
}
* Get top suggestions, those shown when the user clicks in the field but there
* is not enough content to use to suggest. In this case, it will return the row
* and column of the clicked cell clicked.
* @param searchText ignored.
* @param maxSuggestedItems ignored, we only produce 1.
* @return Suggestions to show the user.
*/
@Override
public List<SearchSuggestItem> getTopSuggestions(String searchText,
                             int maxSuggestedItems)
{
    List<SearchSuggestItem> ret = new ArrayList<SearchSuggestItem>();
    ret.add(new SearchSuggestItem(getCell()));
    return ret;
}

private String getCell()
{
    return "Cell: " + getSelectedRow() + "," + getContextCode();
}

/**
* Use the clientId to find the InputText component, the enclosing table and
* thereby the selected row.
* @return selected row, -1 if cannot find for any reason.
*/
private int getSelectedRow()
{
    // There are 2 ways of finding the cell: one is to know which column we are
    // in by being told (I use contextCode here) and determining the row using
    // the table's selected row. However this requires the table to have
    // rowSelection="single" which is likely but not always the case.
    // The second is to use the clientId of the component and a bit of internal
    // knowledge of the encoding of the ids.

    int id1 = getSelectedRowUsingTable();
    int id2 = getSelectedRowUsingClientId();
    if (id1 != id2)
    {
        if (AppsLogger.isEnabled(AppsLogger.WARNING))
        {
            AppsLogger.write(this,
               "TableCellSuggestGroup: getSelectedRow: found differing
rows, row using table selected row: " + id1 + ", by clientId: " + id2;
        }
    }
    // client id is more accurate across different rowSelection attribute settings.
    return id2;
}

private int getSelectedRowUsingTable()
{
    // Method 1:
    UIComponent field = SearchUtils.findComponentByClientId(_searchFieldClientId);
    ...
if (field != null)
{
    RichTable table =
        (RichTable)PatternsUtil.getParentByType(field, RichTable.class);

    if (table != null)
    {
        // find the selected row.
        RowKeySet selectedRowKeys = table.getSelectedRowKeys();
        if (selectedRowKeys != null && selectedRowKeys.size() > 0)
        {
            // This works for Java ArrayLists, but will not work for BC collections.
            Integer row = (Integer)selectedRowKeys.iterator().next();
            return row;
        }
    }
    return -1;
}

private int getSelectedRowUsingClientId()
{
    // Method 2:
    String clientId = getSuggestFieldClientId();
    String[] clientIds = clientId.split(".*");
    if (clientIds.length > 2)
    {
        return Integer.valueOf(clientIds[clientIds.length - 2]);
    }
    return -1;
}

Example of a Prefixed Suggestion Group

package oracle.apps.fnd.searchSuggest;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import oracle.apps.fnd.applcore.patterns.ui.behaviors.searchSuggest.AbstractSearchSuggestGroup;
import oracle.apps.fnd.applcore.patterns.ui.behaviors.searchSuggest.SearchSuggestItem;

/**
 * A search suggest group that shows a fixed set of top suggestions being the:
 * Context Code
 * Object Type
 * with which it was configured (similar to ReflectiveSearchSuggestGroup).
 * Once the user starts typing, it will show suggestions only when a prefix
 * character is typed (in this case a \# character). This character will not be
 * part of the suggestions but will be returned to the field.
 * For testing.
 */
public class PrefixedSearchSuggestGroup
extends AbstractSearchSuggestGroup
{
    /**
     * Unique name of this group. Must be unique across all groups.
     */
public static final String NAME = "prefixedterms";

@Override
public String getName()
{
    return NAME + (getGroupIndex() + 1);
}

@Override
public String getDisplayName()
{
    String overRiddenDisplayName = super.getDisplayName();
    if (overRiddenDisplayName != null)
    {
        return overRiddenDisplayName;
    }
    return "Prefixed Group " + (getGroupIndex() + 1);
}

@Override
public String getIconURL()
{
    String overRiddenIconURL = super.getIconURL();
    if (overRiddenIconURL != null)
    {
        return overRiddenIconURL;
    }
    return "images/app1core/uishell/globalSearch/search_ena.png";
}

@Override
public List<SearchSuggestItem> getItems(String searchText,
                                          int maxSuggestedItems)
{
    List<SearchSuggestItem> ret = new ArrayList<SearchSuggestItem>();
    if (searchText.indexOf("#") == 0)
    {
        for (int i = 1; i <= 5; i++)
        {
            ret.add(new SearchSuggestItem(searchText.substring(1) + "suggestion" + i,
                                           searchText + i));
        }
    }
    return ret;
}

@Override
public List<SearchSuggestItem> getTopSuggestions(String searchText,
                                                  int maxSuggestedItems)
{
    List<SearchSuggestItem> ret = new ArrayList<SearchSuggestItem>();

    ret.add(new SearchSuggestItem(getContextCode()));
    ret.add(new SearchSuggestItem(getObjectType()));

    return ret;
}

@Override
public void init(Map<String, Object> attributes)
15.8 Attachments in Global Search

This section details the design and behavior of the implementation of Attachments in Global Search.

The implementation:

- Shows attachments in the results UI using the same descriptions and tooltips that the attachments component does.
- Allows the user to click an attachment in the results UI and have the attachment open.
- Shows keyword bolding in the attachment name (if any).
- Allows the developer to promote grandchildren attachments if desired. For instance, PO Line attachments are shown with the PO result.
- Works across attachment categories.
- Suppresses secured attachments.
- Picks up searchable view objects with changed attachments and flags for recrawling (but not grandchildren attachments).
- Works with the existing tag crawl functionality.
- Does not allow uploading or deleting of attachments directly from the search UI. This must be done from the user UI that the attachments component is on. This is because attachment requires view links to the Business view objects to which they are attached. This is not available in a search result that may be displayed anywhere in Fusion Applications. However, it is probable that clicking the search result will take the user directly to this UI, so this functionality is only a click away.

15.8.1 Crawl Functionality

Crawl is done using the FndSearchPlugin that can be used either directly, or as a base class to existing plugins.

The plugin replicates the features of the TagSearchPlugin and adds Attachment functionality, as shown in the next image. Developers still can use TagSearchPlugin, but it will not do attachments. Use the FndSearchPlugin to get
both sets of functionality (tags and attachments). The tag setup in the FndSearchPlugin is the same as it is in the TagSearchPlugin.

**Figure 15-50  Search Plugin for Attachments**

There are two arguments to determine how attachments are crawled:

- **ATTACHMENT_VIEW_LINK_ACCESSOR_PATHS**: A comma-separated list of accessor paths to Attachment view objects.

- **ATTACHMENT_INCREMENTAL_CRAWL_ENTITY_NAME**: The name of the attachment entity used in incremental crawls to reverse lookup changed attachments.

**Attribute ATTACHMENT_VIEW_LINK_ACCESSOR_PATHS**

[Optional] This attribute defines the attachments that will be considered as part of the crawl. Without this attribute, the default attachment behavior of ECSF will be unchanged; that is, all direct attachments of the searchable view object will be crawled, but nothing will be shown in the Search UI.

It is possible to have multiple attachment view links from a master view object, and attachments stored against detail view objects in a master-detail relation.

An accessor path is a dot-separated set of accessors to an Attachment view object. The attribute **ATTACHMENT_VIEW_LINK_ACCESSOR_PATHS** is a comma-separated value (CSV) list of these accessor paths. For example, if GsProductAttachmentsAccessor is the view link accessor to the AttachmentVO directly under the Search view object, then this would be the accessor path to these attachments.

Similarly, if GsProductDetails is the accessor to a detail level view object that has an accessor GsProductDetailsAttachmentsAccessor to the attachments, then GsProductDetails.GsProductDetailsAttachmentsAccessor is the accessor path to these attachments.
Any number of these paths can be specified, though it is expected to be small (usually only 1).

As an example, accessors in the Master view object for this would be:

```xml
<ViewLinkAccessor
    Name="GsProductAttachmentsAccessor"
    ViewLink="oracle.apps.model.GsProductAVL"
    Type="oracle.jbo.RowIterator"
    IsUpdateable="false"/>
<ViewLinkAccessor
    Name="GsProductDetails"
    ViewLink="oracle.apps.model.GsProductToDetailsVL"
    Type="oracle.jbo.RowIterator"
    IsUpdateable="false"/>
```

and in the detail view object:

```xml
<ViewLinkAccessor
    Name="GsProductDetailsAttachmentsAccessor"
    ViewLink="oracle.apps.model.GsProductDetailsAVL"
    Type="oracle.jbo.RowIterator"
    IsUpdateable="false"/>
```

When creating master detail relations like this, do not include the accessor in the detail view object. If you do include the accessor in the detail view object, ECSF will try to follow the links from Master > Detail > Master > Detail in a recursive manner until it hits the maximum depth of 7 levels.

In such a case, you will see a crawl error similar to:

```xml
<VODocumentImpl> <assign> Deep-level view objects not supported. VO is: oracle.apps.model.GsProductDetailsVO Level: 7
```

**Attribute ATTACHMENT_INCREMENTAL_CRAWL_ENTITY_NAME**

Because adding an attachment does not change the Last_update_date of the parent view object row, there needs to be a reverse lookup mechanism used to force these additional rows onto the incremental crawl queue.

This attribute is not required. If it is not supplied, attachment additions to a view object row that is otherwise unchanged (and therefore on last_update_date change) will go un-noticed.

This attribute defines the attachment entity name that will be used when determining which searchable view object records need to be flagged for re-crawl because they have had changes to their attachments.

Note that only newly-added attachments and updated attachments (including the title) will be found for re-crawl. There is no mechanism to detect deleted attachments.

**Internal Stored Attribute FND_ATTACHMENT_METADATA**

Attachments found during the crawl will be recorded in the searchable view object in the predefined stored String attribute FND_ATTACHMENT_METADATA. This attribute needs to be defined in the searchable view object definition and flagged as a stored attribute. It should have no data in it because it will be over-written by the Applications Core preIndexProcessor at crawl time. An example of defining an attribute is shown below using a transient attribute.
What is stored in this attribute is an XML structure for all the attachments stored against a searchable object. It does not store the attachment data itself here. The attachment data is stored in the Oracle SES indexed document in an inline for making it available for text search. This example shows four attachments stored against a row:

```xml
<Attachments>
  <Attachment documentId="300100024947778" datatypeCode="FILE" title="Latte.doc" fileName="Latte.doc" />
  <Attachment documentId="300100024947776" datatypeCode="FILE" title="UIShell_dtpreview.png" fileName="UIShell_dtpreview.png" />
  <Attachment documentId="300100024881786" datatypeCode="WEB_PAGE" title="oracle" description="Oracle homepage" fileName="http://oracle.com" />
  <Attachment documentId="300100024881784" datatypeCode="TEXT" title="a quick note" description="a quick note" fileName="a quick note" />
</Attachments>
```

The meaning of the attributes is defined in the below table.

Having these attributes stored in the searchable view object means they are searchable by default, and the UI can use the values to quickly decode the values without costly database look-ups during the rendering of the result.
Table 15-16  Attachment Attributes

<table>
<thead>
<tr>
<th>datatype</th>
<th>Code</th>
<th>Document ID</th>
<th>Title</th>
<th>Description</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>Content server documentId</td>
<td>nvl(title if available, otherwise fileName)</td>
<td>description</td>
<td>fileName</td>
<td></td>
</tr>
<tr>
<td>WEB_PAGE</td>
<td>documentId</td>
<td>nvl(title if available, otherwise URL)</td>
<td>description</td>
<td>URL</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td>documentId</td>
<td>nvl(title if available, otherwise fileName)</td>
<td>description</td>
<td>fileName</td>
<td></td>
</tr>
<tr>
<td>TMURL</td>
<td>Not supported and there will be no attributes stored for it in SES.</td>
<td>nvl(title if available, otherwise URI)</td>
<td>description</td>
<td>URI (not URL)</td>
<td></td>
</tr>
<tr>
<td>FOLDER</td>
<td>Not supported and there will be no attributes stored for it in SES.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Mapping of AttachmentVO attributes into search metadata

**Important:** The XML is supplied without a schema. It is an internal part of Applications Core search and may change at any time without notice.

Attachments found at detail levels are pulled up to the parent level. If this is not required, do not include an accessor path to this level and these attachments will not be crawled.

Attachments in all categories are included. However, if that document entity has security enabled (no matter what the data security rules are), the attachments assigned to that document entity are removed from the indexed document and therefore are not crawled. Their details will not be in the XML.

The following types of attachments are available as links:

- **FILE**
  An attached file is the primary use case.

- **WEB_PAGE**
  A URL, such as http:www.oracle.com. The link will be placed verbatim in the results UI as an af:goLink. Note that ECSF does not crawl Web page attachments.

- **TEXT**
  Text typed in by the user. This is implemented by attachments internally as a hidden file, so is practically the same as FILE.

The following types of attachments are *not* available as links:

- **FOLDER**
  A reference to a folder in UCM. It requires use of UCM folder taskflow.

- **TMURL**
  Topology Manager URI. This is a URI offset to a web application defined in topology. This is a minor use case used in migration and is not supported by search.
Attachment documentId compared to Attached Document Id

documentId is a globally-unique number identifying an attachment. It will return the latest version of the attachment.

Attached DocumentId is a globally-unique number identifying a specific version of an attachment. It will return a specific version of the attachment.

As an example, documentId 100000000001 is for file BRD.doc. It has Attached DocumentId 100000000002 for the initial version of the file, and 100000000003 for the next version of the document.

Global Search will use the documentId and always download the latest version of the file.

15.8.2 Show Attachments

Attachments will be shown on a line below the "Other Actions" in the Search Result. That is the lowest part of a result item.

The text of the link will be the Attachment title, which generally defaults to the filename in the Attachments UI, but may be overridden by the user when the attachment is uploaded.

The tool tip of the attachment will be the attachment description entered by the user. If this is not present, the title value will be used.

Attachments will be sorted alphabetically.

All information is available in the stored attribute of the Search view object to render attachment links in the UI with no additional lookup work, and therefore there is no significant performance overhead.

This image shows documents attached to a search result and the tooltip.

Figure 15-52 Documents Attached to a Search Result

15.8.3 Open Attachments

Clicking an attachment link, shown in this figure, will open the content for the user. The browser settings should determine how this is done and should be in line with what the Attachment component does.
For example, clicking a JPG file attachment link usually will open it directly in a browser tab, while clicking a .doc file link usually will open it in Microsoft Word.

15.8.4 Implement the Attachment Plugin

There are two cases to consider:

- The searchable view object has no plugin currently configured: Simply use the Applications Core plugin.
- The searchable view object has a plugin already: Set up the existing plugin to inherit the Applications Core plugin.

15.8.4.1 Use the Applications Core Plugin

To use the existing Applications Core plugin, follow these steps:

1. In the ECSF Search view object design time, add a Search Plugin to the Applications Core class `oracle.apps.fnd.applcore.search.plugin.FndSearchPlugin`.

2. Configure attachments that will be crawled.

   Add a parameter with a comma-separated list of accessor paths with key `ATTACHMENT_VIEW_LINK_ACCESSOR_PATHS`.

3. Configure searchable view object incremental crawl detection of searchable view objects with new attachments.

   Optionally, add an attribute with key `INCREMENTAL_CRAWL_ATTACHMENT_ENTITY_NAME` and value of the Attachment Entity Name (the value you enter in the Attachment View Link wizard, such as `GS_PRODUCTS`).

   This will add searchable view objects to the crawl list when new attachments are added. This is useful as the searchable view object last_update_date value is not updated when an attachment is added, and the document will not be detected as needing a re-crawl. This will flag the individual searchable view object row to be recrawled. Note that it is not possible to detect changed attachments this way at the detail level (for example a purchase order line's attachments), because the reverse
lookup would obtain the primary key of the line item and not the searchable view object.

**15.8.4.2 Enhance Existing Plugin**

If you have your own plugin, extend your plugin using the following steps:

1. Extend the Applications Core plugin and call super() on the methods that implement attachment functionality. Use the same parameters to the plugin as described above.

   ```java
   public class AppsSearchPlugin
       extends FndSearchPlugin
       implements PreIndexProcessor,
                   ChangeListener
   {
       /**
        * Hook into the pre index process of the crawl so we can annotate the SVO
        * with apps specific information such as tags and attachment metadata.
        * @param ctx Search Context.
        * @param docs Documents to iterate and find tag children for.
        */
       public void preIndexProcess(SearchContext ctx,
                                   List<IndexableDocument> docs)
       {
           super.preIndexProcess(ctx, docs);
           // custom apps functionality ...
       }

       /**
        * Get the change list for crawling.
        * @param ctx Search Context.
        * @param changeType the type of the change requested. Must be one of following
        * <ul>
        * <li>ChangeListener.UPDATE<li>
        * <li>ChangeListener.INSERT<li>
        * <li>ChangeListener.DELETE<li>
        * </ul>
        * @return an iterator of a list of primaryKeys. Returns an empty iterator if
        * nothing changed.
        */
       public Iterator getChangeList(SearchContext ctx, String changeType)
       {
           Iterator ret = super.getChangeList(ctx, changeType);
           // custom apps functionality ...
           return ret;
       }
   }
   ```

2. Add the transient stored attribute FND_ATTACHMENT_METADATA.

   Add a transient String attribute to the searchable view object named FND_ATTACHMENT_METADATA to store Applications Core attribute metadata. Flag this attribute as stored in the searchable view object definition.

**15.9 Support Global Search Alternate Words for the Cloud**

To support Cloud customers, a Fusion Applications user interface allows upload and download of alternate search words to Oracle Secure Enterprise Search.
Oracle SES is the standalone database and Oracle WebLogic Server applications server used for search. It provides a store of alternate words and web service endpoints for searching and administration.

**Overview**

A new taskflow provides maintenance of Oracle SES alternate words.

It uses APIs to determine the Oracle SES Administration service endpoint, username and password, and uses the Oracle SES web service stubs available in the Oracle Enterprise Crawl and Search Framework shared library.

The setup taskflow is available in the `GlobalSearch.jws` application.

FndSetup will host the taskflow, as it does for all setup flows. It already has the "ECSF Runtime Server" library in the `weblogic-application.xml` file:

```xml
<library-ref>
  <library-name>oracle.ecsf</library-name>
</library-ref>
```

The taskflow will talk to Oracle SES directly, instead of to ECSF search that federates through search servers that talk to Oracle SES. There is no need for federation because there is only one Oracle SES for Fusion Applications.

The setup flow dialog appears similar to this example:

**Figure 15-54   Example of the Alternate Words Maintenance Taskflow Dialog**

<table>
<thead>
<tr>
<th>Welcome</th>
<th>Maintain SES Alternate Words x</th>
</tr>
</thead>
</table>

**Manage Alternate Words**

| Filter: | | |
|---------| | |

| Actions | View | | | |
|---------|------| | | |

<table>
<thead>
<tr>
<th>User Input Keyword</th>
<th>Alternate Keyword</th>
<th>Automatically Search Both</th>
</tr>
</thead>
<tbody>
<tr>
<td>bea</td>
<td>oracle</td>
<td>✓</td>
</tr>
<tr>
<td>cloud</td>
<td>oracle</td>
<td>✓</td>
</tr>
<tr>
<td>database</td>
<td>oracle</td>
<td>✓</td>
</tr>
<tr>
<td>oracle</td>
<td>oracle</td>
<td>✓</td>
</tr>
</tbody>
</table>

The taskflow has one parameter:

<table>
<thead>
<tr>
<th>Name</th>
<th>Optional</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pageTitle</td>
<td>Yes</td>
<td>Title to be shown at the top of the page. The default is &quot;Manage Alternate Words&quot;</td>
</tr>
</tbody>
</table>
The dialog uses these buttons:

- **Add** - Add a new row above the selected one (or row 0 if none are selected).
- **Duplicate** - Duplicate the existing row.
- **Edit** - Edit the current row (all fields).
- **Delete** - Delete the current row.

Note that a row is identified by the "User Input Keyword" and "Alternate Keyword."
A user can edit any field on the row, including the two key fields. This will delete the old record and add the new one.

### 15.9.1 Look Up the SES Endpoint (ECSF APIs)

ECSF/SES does not use the connections architecture (connections.xml) to determine SES endpoints, which are saved in ECSF tables.

This code, which mimics ECSF directly, is used to look up the endpoint. Because there is only one SES per Fusion Applications instance, the first configured endpoint will be used.

```java
private MetaEngineInstance getActiveLocalInstance()
{
    MetaEngineInstance ret = null;
    List<MetaEngineInstance> allECSFComponents = MetaDataManager.getEngineInstances(3);
    for (MetaEngineInstance localInstance : allECSFComponents)
    {
        Map params = localInstance.getParameters();
        String sesAdminUsername = (String)params.get("SES_ADMIN_USERNAME");
        String sesAdminService = (String)params.get("SES_ADMIN_SERVICE");
        String isActive = (String)params.get("IS_ACTIVE");
        if (sesAdminUsername != null && sesAdminUsername.indexOf("[CHANGE_ME]") == -1
            && sesAdminService != null && sesAdminService.startsWith("http"))
        {
            ret = localInstance;
            break;
        }
    }
    return ret;
}
```

The endpoint, username and password will be used to create the Administration port connection (see below).

If no endpoint can be found, an error will be shown - see the error messages section below for details on the error messages that may be shown.
15.9.2 SES Web Service Calls

There are four web service calls used by Global Search Alternate Words for the Cloud.

Administration Port Connection

```java
private AdminPortType getStatefulWebServiceClient(
    String webServiceURL,
    String userName,
    String password) throws Exception
{
    AdminService adminService = new AdminService(
        new URL( webServiceURL ),
        new QName(
            "http://search.oracle.com/Admin",
            "AdminService"
        )
    )
    
    AdminPortType adminPort = adminService.getAdmin();

    // Tell client proxy to maintain HTTP session for stateful behavior
    ((BindingProvider)adminPort).getRequestContext().put(
        BindingProvider.SESSION_MAINTAIN_PROPERTY, true
    )

    // Create credentials argument
    Credentials credentials = new Credentials();
    credentials.setUserName( userName );
    credentials.setPassword( password );

    adminPort.login( credentials, getLanguage() );

    return adminPort;
}
```

Alternate Word Retrieval

This example shows how to retrieve an alternate word.

```java
private List<AlternateWord> getAltWords(AdminPortType adminPort) throws Exception
{
    ObjectOutput oo = adminPort.exportAll(
        "altWord",
        null,
        null, // "password",
        null, // credentials,
        null,
```

Figure 15-55  Endpoint Not Found Message
Alternate Word Deletion

This example shows how to delete an alternate word.

```java
private void removeAltWords(AdminPortType adminPort,
                            List<AlternateWord> altWords)
    throws Exception {
    List<ObjectKey> objectKeys = new ArrayList<ObjectKey>();
    for (AlternateWord word : altWords) {
        // Need to use the original key values in case the user has updated, then
        // deleted the row. Note that do not need to worry about new rows as these
        // will never be in the deleted list.
        objectKeys.add(createKeyPairs(word.getOrigKeyword(), word.getOrigAltWord()));
    }

    List<OperationControl> ops = toOperationControlList("IGNORE_NOT_FOUND", "true");
    adminPort.deleteList("altWord", objectKeys, null, // credentials,
                        ops, getLanguage());
}
```

Alternate Word Insertion

This example shows how to insert an alternate word.

```java
private void createAltWords(AdminPortType adminPort,
                            List<AlternateWord> altWords)
    throws Exception {
    StringBuilder sb = new StringBuilder(
    "<?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE search:config PUBLIC "http://xmlns.oracle.com/search/search.dtd">
    " +
    "<search:altWords>");
    for (AlternateWord word : altWords) {
        sb.append(toAltWordXML(word.getKeyword(), word.getAltWord(),
                                word.isAutoExpand()));
    }
    sb.append("</search:altWords>" +
              "</search:config>");

    List<OperationControl> ops = toOperationControlList("DUPE_METHOD", "overwrite");
    adminPort.createAll("altWord", sb.toString(), null, //"password",
                        null, // credentials,
                        ops, getLanguage());
}
```
null, // credentials,
null,
ops,
getLanguage()
);}

15.9.3 Alternate Words Error Messages

This table shows error messages that may be shown to the user for setup or web service communications issues.

<table>
<thead>
<tr>
<th>Message</th>
<th>Where Shown</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Enterprise Search is not configured in this environment.</td>
<td>When retrieving the list of alternate words when the taskflow starts.</td>
<td>The ECSF metadata that configures SES is not available or is not complete. The SES web service endpoint cannot be determined. It is likely Global Search is also not available.</td>
</tr>
<tr>
<td>Secure Enterprise Search is not responding.</td>
<td>When retrieving the list of alternate words when the taskflow starts, or saving after the &quot;Save and Close&quot; button is pressed.</td>
<td>The SES endpoint cannot be contacted.</td>
</tr>
<tr>
<td>Cannot retrieve the list of Alternate Words from Secure Enterprise Search.</td>
<td>When retrieving the list of alternate words when the taskflow starts</td>
<td>The SES endpoint cannot be contacted, but the list of alternate words cannot be retrieved from it.</td>
</tr>
<tr>
<td>The following Alternate Words already exist for the User Keyword. Delete the duplicates:</td>
<td>When saving alternate words.</td>
<td>Within the list of alternate words there are duplicate keyword/altword pairs.</td>
</tr>
</tbody>
</table>

15.10 Implement List Search

List Search, sometimes known as "local search," is for implementing search functions on a single data source (also known as a View Object).

It comprises a keyword search box, autosuggest recent items and saved searches, and a collapsible Filters panel that uses ADF Query. Use the full List Search component when users need keyword search augmented by fielded search and saved searches. This image shows an overview of List Search.
A new declarative component supports List Search. It is available from the Applications Core component gallery and from the Data Control palette for drag-and-drop of View Criteria to the page.

A running List Search would resemble this image.
Figure 15-57  List Search Declarative Component with Suggestions

This image shows:

- The omnibox is labelled "Search" with shadow text.
- A set of suggestions (note the UI is pre-Rel 10 so has changed a bit),
- Recent items from the existing recentitems suggest group,
- Three saved searches - "All People", "Analyst" and "ElizabethDirects."
CSS Styles

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FndListSearch</td>
<td>Style on the root-most af:panelStretchLayout that makes up fnd:listSearch</td>
</tr>
<tr>
<td>FndListSearchQuery</td>
<td>Style set on the af:query within fnd:listSearch</td>
</tr>
<tr>
<td>FndListSearchClearLink</td>
<td>Style set on the clear link (the ‘X’) on the right of the search field</td>
</tr>
</tbody>
</table>

Declarative Components

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Evaluated Type</th>
<th>Values</th>
<th>Default</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>n/a</td>
<td>Yes</td>
<td></td>
<td>Generated for DC automatically by ADF.</td>
</tr>
<tr>
<td>rendered</td>
<td>boolean</td>
<td>true/false</td>
<td>true</td>
<td>No</td>
<td>Generated for DC automatically by ADF.</td>
</tr>
<tr>
<td>binding</td>
<td>RichDeclarativeComponent</td>
<td>n/a</td>
<td>n/a</td>
<td>No</td>
<td>Generated for DC automatically by ADF.</td>
</tr>
<tr>
<td>keywordSavedSearch</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>Yes</td>
<td>Name of the search used for keyword searches. Must correspond to a View Criteria name (the code name, not the translated name). This View Criteria will be run when the user types in free-form search terms.</td>
</tr>
<tr>
<td>keywordSearchField</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>Yes</td>
<td>Name of the field in the keywordSavedSearch to place the search text when performing a keyword search. Must correspond to a text field in the keywordSavedSearch View Criteria whose default value is not set. This field will be assigned the free-form search terms entered by the user, and should be at least 30 characters in length.</td>
</tr>
<tr>
<td>Attribute Name</td>
<td>Evaluated Type</td>
<td>Values</td>
<td>Default</td>
<td>Required</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------</td>
<td>--------</td>
<td>---------</td>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>suggestGroups</td>
<td>java.lang.List</td>
<td>n/a</td>
<td>null</td>
<td>No</td>
<td>The suggest groups to show in the omnibox. This is a List of SearchSuggestGroupOption and will be used instead of suggestGroup1 .. 8 and the contextCode and objectType if available. Some implementations may find it easier to define the groups in a managed bean this way.</td>
</tr>
<tr>
<td>suggestGroup1 .. 8</td>
<td>String</td>
<td>n/a</td>
<td>null</td>
<td>No</td>
<td>Individual suggestion Groups to be shown in the order defined.</td>
</tr>
<tr>
<td>contextCode1 .. 8</td>
<td>String</td>
<td>n/a</td>
<td>null</td>
<td>No</td>
<td>Context codes for the appropriately numbered group.</td>
</tr>
<tr>
<td>objectType1 .. 8</td>
<td>String</td>
<td>n/a</td>
<td>null</td>
<td>No</td>
<td>Object types for the appropriately numbered group.</td>
</tr>
<tr>
<td>queryBindingId</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>Yes</td>
<td>The id of the af:query search region, for example in the below pageDef entry it would be &quot;AllAttributesQuery.&quot;</td>
</tr>
</tbody>
</table>

```
Implement List Search
<searchRegion
Criteria="AllAttributes"
    Customizer="...
    Binds="PeopleIterator"

id="AllAttributesQuery"/>
```
<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Evaluated Type</th>
<th>Values</th>
<th>Default</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resultBindingId</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>Yes</td>
<td>The id of the results binding, for example in the below pageDef entry is would be &quot;GsEmp1&quot;. This is used to get the estimated result count to update the keywords list.</td>
</tr>
<tr>
<td>queryListener</td>
<td>javax.Express Language.MethodExpression</td>
<td>n/a</td>
<td>null</td>
<td>Yes</td>
<td>Required if you wish to customize the query criteria - see <a href="http://www.oracle.com/technetwork/developer-tools/ADF/learnmore/85-querycomponent-fieldvalidation-427197.pdf">http://www.oracle.com/technetwork/developer-tools/ADF/learnmore/85-querycomponent-fieldvalidation-427197.pdf</a> Bound to the ADF queryListener value in the pageDef, for example: #{bindings.AllAttributesQuery.processQuery}</td>
</tr>
<tr>
<td>resultComponentId</td>
<td>String</td>
<td>n/a</td>
<td>n/a</td>
<td>No</td>
<td>Id of result for af:query to ppr.</td>
</tr>
<tr>
<td>searchFieldLabel</td>
<td>String</td>
<td>n/a</td>
<td>&quot;Search&quot;</td>
<td>No</td>
<td>Label of the search field.</td>
</tr>
<tr>
<td>searchFieldPlaceholder</td>
<td>String</td>
<td>n/a</td>
<td>null</td>
<td>No</td>
<td>The shadow text (placeholder text) of the Search field.</td>
</tr>
<tr>
<td>Attribute Name</td>
<td>Evaluated Type</td>
<td>Values</td>
<td>Default</td>
<td>Required</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
<td>--------</td>
<td>---------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>searchFieldTextSelection</td>
<td>String</td>
<td>default</td>
<td>default</td>
<td>No</td>
<td>How is text selection handled in the field. Options are 'default' which will use the inputText field default selection which is typically selection of all text on tab into the field (likely browser dependent) and 'click' which will select all text in the field when the field is first clicked in with the mouse.</td>
</tr>
<tr>
<td>suggestGroupHeadings</td>
<td>String</td>
<td>on/off</td>
<td>on</td>
<td>No</td>
<td>Whether group heading are shown in the search field autosuggest.</td>
</tr>
<tr>
<td>suggestGroupIcons</td>
<td>String</td>
<td>on/off</td>
<td>off</td>
<td>No</td>
<td>Whether group icons are shown in the search field autosuggest.</td>
</tr>
<tr>
<td>suggestNoMatchMessage</td>
<td>String</td>
<td>n/a</td>
<td>null</td>
<td>No</td>
<td>The message shown when there is no match to the user's suggestions.</td>
</tr>
<tr>
<td>suggestTopSuggestions</td>
<td>String</td>
<td>on/off</td>
<td>on</td>
<td>No</td>
<td>Whether top suggestions are shown when the user first clicks in the field.</td>
</tr>
<tr>
<td>suggestMinCharCount</td>
<td>Integer</td>
<td>n/a</td>
<td>1</td>
<td>No</td>
<td>The minimum number of characters before suggestions are shown. Before this number top suggestions may be shown.</td>
</tr>
<tr>
<td>suggestMaxSuggestedItems</td>
<td>Integer</td>
<td>n/a</td>
<td>15</td>
<td>No</td>
<td>The maximum number of suggestions shown to the user.</td>
</tr>
<tr>
<td>suggestSubmitKeywordOnSelection</td>
<td>Boolean</td>
<td>n/a</td>
<td>true</td>
<td>No</td>
<td>Does the selection of a keyword from the suggestion list also submit a search?</td>
</tr>
<tr>
<td>splitterPosition</td>
<td>Integer</td>
<td>n/a</td>
<td>300</td>
<td>No</td>
<td>The default splitter position.</td>
</tr>
<tr>
<td>splitterCollapsed</td>
<td>Boolean</td>
<td>true / false</td>
<td>true</td>
<td>No</td>
<td>The default splitter collapse state.</td>
</tr>
</tbody>
</table>
Facets

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>results</td>
<td>Results area containing Design Time-generated output and modifiable by the developer</td>
</tr>
</tbody>
</table>

15.10.1 How to Implement List Search at Design Time

The developer develops View Criteria using normal BC DT. The default BC View Object View Criteria UI in JDeveloper can be used, although many options need to be disabled or simplified.

View Criteria

There should be one View Criteria that will be later designated the "Default Saved Search." This saved search is run when the `fnd:listSearch` component is first run. It is important that this criteria is well tuned and runs quickly because it directly affects the initial rendering of the `fnd:listSearch` component.

There should be another saved search that later will be designated the "Keyword Saved Search." This saved search is used whenever the user enters keywords and clicks Search. It must have a Text field of at least 30 characters in width that is tuned to provide fast query responses (30 characters is the width of the search field). If the field is shorter than this, the keywords will be truncated.

Developers should not use only one View Criteria to perform as both the default and keyword saved searches.

Once generated, these saved searches will display in the JDeveloper Data Controls Palette as "Named Criteria," also known to `af:query` as system saved searches.

The development of these two View Criteria is mandatory before the `fnd:listSearch` component can be configured.

An example Keyword Saved Search View Criteria is shown below using the BC View Object UI in JDeveloper.
Click the Criteria Definition tab.
The **Criteria Name** (not the Display Name) is used as the `keywordSavedSearch` attribute later in the `fnd:listSearch` component.

Click the Criteria UI Hints tab, then the Custom Operators tab.
Figure 15-60  Custom Operators Tab

**Component**

Developers will drag and drop the VO View Criteria they wish to be the Design Time default saved search (Data First DT). The developer will drag and drop the listview component from the Component Palette (Component First DT).

The user will be able to change the default saved search later by personalizing the saved searches and selecting another default, but this will be the initial default for all users.

The Create List Search wizard starts and displays the Specify data source and criteria dialog.

Turn off Custom Operators.
When dragged-and-dropped from the Data Controls Palette, this is the View Criteria that is dropped.

The user must select a default saved search to continue. The "All Queriable Attributes" saved search produced by ADF cannot be selected.

Click Next to specify the Keyword Saved Search.
Figure 15-62  Keyword Saved Search

Select the keyword saved search from the list of all View Criteria, and the keyword field from this View Criteria.

The developer designated saved search is the view criteria previously selected. It can be changed by the user at runtime.

Default Saved Search:  DefaultCriteria

Select the criteria to be used as the keyword saved search from the list of named view criteria, and the field that will be used to query against.

Keyword Saved Search:  DefaultCriteria
Keyword Field:  Ename

Select the keyword saved search from the list of all View Criteria, and the keyword field from this View Criteria.

The "All Queriable Attributes" criteria produced by ADF will not be available for selection.

Click Next to specify the Search Field Properties.
Set the Search Field Label and Search Field Placeholder.

Select Suggestion Groups and their contextCode and objectTypes. Three suggestion groups will be available:

- **Search Terms** - A new suggest group for keywords, and a contextCode/objectType to save these under.

- The contextCode and objectType will allow the keywords to be specific to the list search view object.

- **Recent items** - The existing recent items suggest group with a contextCode/objectType.

- **List Saved Searches** - The list search names, this will be the same values seen in the af:query saved search drop down.
Click **Next** to specify the type of results.

**Figure 15-64  Select Results Type**

The developer can select from a number of result types:

- **Card and List View** - This will invoke the Card and List Layout wizard.
- **Applications Table** - This will invoke the Applications Table wizard.
- **ADF List View - panelFormLayout** - This will allow results in a `listView` using a `panelFormLayout` to format the results in the List View row.
- **ADF List View - Grid Templates** - Allow the use of the Quick Grid wizard to configure data in a `listView`.
- **ADF List View - Detailed Grid** - Allow the use of the Detail Grid wizard to configure data in a `listView`.
- **ADF Table** - Use an `af:table` as the results.

This selection determines the rest of the steps except for the final (summary) step.

**Results Card and Listview**

This image shows the first of the four wizard dialogs used when the Card and Listview option was selected in the previous dialog.

**Applications Table**

If Applications Table was selected, another three wizard dialogs are displayed. See [Adding Applications Table Components Using the Applications Table Wizard](#) for
descriptions of the Set Row Selection and Layout Columns, the Select the table CRUD patterns, and the Set Table Features dialogs.

After these dialogs, click Next to display the Summary. This example shows the Summary information for the Card and Listview selection. The information for the Application Table selection is almost identical.

Click Finish to create the code, or Back to make changes.

These images show examples of the code that is created.
Implement List Search

Implement Search Functions in the UI Shell
Implement List Search

```xml
<pageDefinition xmlns="http://xmlns.oracle.com/adfm/uimodel" Version="11.1.1.67.54" id="untitled1PageDef" Package="oracle.apps.view.pageDefs">
  <parameters/>
  <executables>
    <variableIterator id="variables"/>
    <iterator Binds="GsEmp1" RangeSize="25"
      DataControl="AppModuleAMDataControl1" id="GsEmp1Iterator"/>
    <searchRegion Criteria="DefaultCriteria"
      Customizer="oracle.jbo.ui.cli.binding.JUSSearchBindingCustomizer"
      Binds="GsEmp1Iterator id=DefaultCriteriaQuery'/">
      <bindings>
        <tree IterBinding="GsEmp1Iterator" id="GsEmp1">
          <nodeDefinition DefName="applcore.ecsf.test.model.GsEmpVO" Name="GsEmp10">
            <AttrNames>
            <Item Value="Empno"/>
            <Item Value="Ename"/>
            <Item Value="Job"/>
            <Item Value="Mgr"/>
            <Item Value="Hiredate"/>
            <Item Value="Sal"/>
            <Item Value="Comm"/>
            <Item Value="Deptno"/>
            <Item Value="Attachment"/>
            <Item Value="StateId"/>
            <Item Value="CityId"/>
            <Item Value="Lang"/>
            <Item Value="AttachmentId"/>
            <Item Value="Binattachment"/>
            <Item Value="Gender"/>
            <Item Value="CountyId"/>
            <Item Value="LastUpdateDate"/>
            <Item Value="LastUpdateLogin"/>
            <Item Value="LastUpdatedBy"/>
            <Item Value="CreatedBy"/>
            <Item Value="CreationDate"/>
            </AttrNames>
          </nodeDefinition>
        </tree>
      </bindings>
    </searchRegion>
  </executables>
</pageDefinition>
```
This chapter discusses additional functions, such as the Navigate API and how to implement the Home Page UI, that are included in the UI Shell page template used to build web pages.

This chapter includes the following sections:

- Introducing the Navigate API
- Warning of Pending Changes in the UI Shell
- Implementing the Oracle Fusion Home Page UI
- Using the Single Object Context Workarea
- Implementing the Third-Party Component Area
- Developing an Activity Guide Client Application with the UI Shell
- Implementing Touch Device Support in the UI Shell
- Implementing the Translation Editor
- About the Create Page Wizard
- Troubleshooting UI Shell Issues

For more information about the features, see:

- Getting Started with Your Web Interface
- Implementing the UI Shell
- Implementing Search Functions in the UI Shell
- Implementing UIs in JDeveloper with Application Tables_Trees and Tree Tables
- Implementing Applications Panels_Master-Detail_Hover_ and Dialog Details

16.1 Introducing the Navigate API

You can create a link in the task flow to open a different UI Shell page. Because navigation can occur across different web applications, a single consistent API performs browser redirect. See Table 16-1. This API is exposed as the FndUIShellController.navigate Data Control method.
Note:
When opening a new window, whenever possible, you should use the Navigate API instead of using the ADF Controller sub-flow calls. This is because the UI Shell has no information about sub flows. For instance, if you perform a search that returns clickable results and use sub flows to display the results, and if you then choose to save the page to your Favorites, the search page is saved; not the results page you want. If you use the Navigate API, however, the UI Shell knows about the opened results page and saving the page to your Favorites works as you expect.

16.1.1 How to Use the Navigate API Data Control Method

Drag and drop the Data Control method on page fragments to create links to start navigation.

1. Expand the Data Controls and select the navigate item, as shown in Figure 16-1.

![Figure 16-1 Selecting navigate from Data Controls](image)

2. Drag navigate and drop it onto the page fragment. When you do, the Applications Context menu shown in Figure 16-2 is displayed so you can choose one of the options.

![Figure 16-2 Selecting a Navigate Option from the Applications Context Menu](image)

Example 16-1 Navigating from One Work Space to Another

```java
/**
 * Navigate from one work space to another. If the current view id is the
 * same as the viewId parameter, and if navTaskFlow is null, the current page
 * will be refreshed. If the current view id is the same as the viewId
 * parameter and if navTaskFlowId is not null, the current page will not
 * be refreshed. The Navigate API will delegate to openMainTask API to open the
 */
```
* task flow in the main area. This can be overridden by passing in *
* forcePageRefresh=true in pageParametersList to force the page refresh so *
* that openMainTask would not be used.
*
* @param viewId viewId of the target workspace
* @param webApp Deployed Module Name of the target workspace.
*   It only must be set when the target workspace is in a different *
*   deployed module than the origin workspace. When this is null, it means *
*   the target workspace is in the same deployed module of the origin *
*   workspace. The webApp attribute is used to look up the host and port of *
*   the associated Workarea or Dashboard from the ASK deployment *
*   tables. These tables are populated at deployment time through *
*   Oracle Fusion Functional Setup Manager tasks.
* @param pageParametersList Parameters list for the page. This is a semicolon *
*   delimited String of name value pairs. For example, *
*   "param1=value1;param2=value2"
*   If the Expression Language expression evaluates to an Object, *
*   toString value of that Object will be passed as the value of *
*   the parameter.
*   When forcePageRefresh=true is added to the pageParametersList, *
*   the current page will be refreshed even if navTaskFlowId *
*   is not null and current view id is the same as viewId *
*   parameter value.
* @param navTaskFlowId Id of the taskFlow to open in the target workspace
* @param navTaskKeyList Key list to pass in to the task flow to open in the *
*   target workspace. This is a semicolon delimited *
*   keys or key-value pairs. For example, *
*   "key1;key2=value2"
* @param navTaskParametersList Parameters list to pass in to the task flow to open *
*   in the target workspace. This is a semicolon delimited String *
*   of name value pairs. For example, *
*   "param1=value1;param2=value2."
*   When loadDependentFlow=true is added to the *
*   navTaskParametersList, the task flow specified in *
*   navTaskFlowId will be loaded as dependent flow.
* @param navTaskLabel Label for the task flow to open in the target workspace.
* @param methodParameters Construct FndMethodParameters object for setting the *
*   width of the contextual area, and/or setting the disclosed *
*   state of the contextual area.
* @throws IOException
*/

public FndMethodParameters navigate(java.lang.String viewId,
        java.lang.String webApp,
        java.lang.String pageParametersList,
        java.lang.String navTaskFlowId,
        java.lang.String navTaskKeyList,
        java.lang.String navTaskParametersList,
        java.lang.String navTaskLabel,
        FndMethodParameters methodParameters)
        throws java.io.IOException

Example 16-2 Setting contextualAreaWidth and contextualAreaCollapsed Parameters

FndMethodParameters methodParameters = new FndMethodParameters();
methodParameters.setContextualAreaCollapsed(Boolean.FALSE);
methodParameters.setContextualAreaWidth(200);

You can specify a task flow to load on the target page. Page level parameters also can be specified.
### Navigate API Parameters

<table>
<thead>
<tr>
<th>Navigate API Parameter</th>
<th>Attribute Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>viewId</td>
<td>navigateViewId</td>
</tr>
<tr>
<td>webApp</td>
<td>Attribute used to look up the host and port of the associated Workarea or Dashboard from the Oracle Fusion Applications Functional Core (ASK) deployment tables. These tables are populated at deployment time through Oracle Fusion Functional Setup Manager tasks. You need to pass the deployed module name to the webApp parameter.</td>
</tr>
<tr>
<td>pageParametersList</td>
<td>navigateParamsList</td>
</tr>
<tr>
<td>navTaskFlowId</td>
<td>Equates to the openMainTask taskFlowId parameter. This is the Id of the task flow to be loaded. The task flow Id is a concatenation of the file location for the task flow definition, and the task flow name. For example: /WEB-INF/MyTaskFlow.xml#MyTaskFlow</td>
</tr>
<tr>
<td>navTaskKeyList</td>
<td>Equates to the openMainTask keyList parameter. Important: navTaskKeyList is used with the task flow Id to identify the target tab in the main area. As such, navTaskKeyList is applicable only in dynamic tabs mode, and is ignored in no-tabs mode. navTaskKeyList provides a way to identify a task flow instance. When reuseInstance is true, use the specified navTaskKeyList in addition to the task flow Id to identify the target tab. The navTaskKeyList parameter has been implemented for the following FndUIShellController data control methods: openMainTask discloseRegionalTask collapseRegionalTask navigate openSubTask In dynamic tabs mode, when looking for a match of an existing tab, these APIs will first look for any instances of the task flow that is already open, which has the same task flow Id as the one passed into them as the parameter. In addition, it will compare the navTaskKeyList values, such that the existing task flow will be picked only if its navTaskKeyList values match the ones specified in the navTaskKeyList parameter. It does not matter if the task flow parameters are the same or different. If the keyList is not set in the menu metadata, you can reuse a tab only if you pass in a null keyList.</td>
</tr>
</tbody>
</table>
Table 16-1  (Cont.) Navigate API Parameters

<table>
<thead>
<tr>
<th>Navigate API Parameter</th>
<th>Attribute Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>navTaskParametersList</td>
<td>You can load the task flow specified in the navTaskFlowId parameter of the Navigate API as a dependent flow if the destination page is in no-tab mode by adding a name-value pair of &quot;loadDependentFlow=true&quot; to navTaskParametersList. For example: &quot;customerId=123;loadDependentFlow=true&quot;</td>
</tr>
<tr>
<td>navTaskLabel</td>
<td>Equates to the openMainTask label parameter. Note: When passing parameters, do not leave the navTaskLabel field null. This is the label that would appear in the tab header when in a tabs page. Even if you are in a no-tabs page (see Supporting No-Tab Work Areas), do not leave it blank because this label will be used in other ways, such as Add to Favorites, or when the system tracks the Recent Items.</td>
</tr>
</tbody>
</table>

Certain parameters, summarized in Table 16-2, can be passed using the Navigate API's argument list.

Table 16-2  Parameters Passed Using Navigate API Arguments

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Passed Using</th>
<th>Description</th>
</tr>
</thead>
</table>
| forcePageRefresh     | pageParametersList   | Default: false
The Navigate API is used to navigate from one work space to another. If the current view Id is the same as the viewId parameter, and if navTaskFlow is null, the current page will be refreshed. If the current view Id is the same as the viewId parameter, and if navTaskFlowId is not null, the current page will not be refreshed, and the Navigate API will delegate to the openMainTask API to open the task flow in the main area. This can be overridden by passing in forcePageRefresh=true in pageParametersList to force the page refresh so that openMainTask would not be used.
Example:
pageParametersList = "forcePageRefresh=true;param2=value2"
Details:
pageParametersList is a semicolon delimited string of name-value pairs. |
Table 16-2  (Cont.) Parameters Passed Using Navigate API Arguments

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Passed Using</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContextualAreaCollapse d</td>
<td>methodParameters</td>
<td>Default: false</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FndMethodParameters methodParameters = new</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FndMethodParameters();</td>
</tr>
<tr>
<td></td>
<td></td>
<td>methodParameters.setContextualAreaCollapsed(Boolean.TRUE);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>methodParameters.setContextualAreaWidth(200);</td>
</tr>
<tr>
<td>contextualAreaWidth</td>
<td>methodParameters</td>
<td>Default: 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FndMethodParameters methodParameters = new</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FndMethodParameters();</td>
</tr>
<tr>
<td></td>
<td></td>
<td>methodParameters.setContextualAreaCollapsed(Boolean.TRUE);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>methodParameters.setContextualAreaWidth(200);</td>
</tr>
<tr>
<td>loadDependentFlow</td>
<td>navTaskParametersList</td>
<td>Default: false</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>navTaskParametersList = &quot;loadDependentFlow=true;param2=value2 &quot;;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note that the value for loadDependentFlow is case sensitive.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Details:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>navTaskParametersList is a parameter list to pass in to the task flow to open</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in the target workspace. This is a semicolon delimited string of name-value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pairs.</td>
</tr>
<tr>
<td>contextCode</td>
<td>methodParameters</td>
<td>This value should be passed only when the navTaskFlowId input parameter has a value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FndMethodParameters methodParameters = new</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FndMethodParameters();</td>
</tr>
<tr>
<td></td>
<td></td>
<td>methodParameters.setContextCode(&quot;CRM&quot;);</td>
</tr>
</tbody>
</table>
Table 16-2  (Cont.) Parameters Passed Using Navigate API Arguments

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Passed Using</th>
<th>Description</th>
</tr>
</thead>
</table>
| objectType     | methodParameters | This value should be passed only when the navTaskFlowId input parameter has a value.  
Example: 
FndMethodParameters methodParameters = new FndMethodParameters();
methodParameters.setContextCode("CRM");
methodParameters.setObjectType("Opportunity"); |
| objectName     | methodParameters | This value should be passed only when the navTaskFlowId input parameter has a value.  
Example: 
FndMethodParameters methodParameters = new FndMethodParameters();
methodParameters.setContextCode("CRM");
methodParameters.setObjectType("Opportunity");
methodParameters.setObjectName("Opportunity 123"); |

Note:
When passing parameters, do not leave the label field null. This is the label that would appear in the tab header when in a tabs page. Even if you are in a no-tabs page (see Supporting No-Tab Work Areas), do not leave it blank because this label will be used in other ways, such as Add to Favorites, or when the system tracks the Recent Items.

The signature and Javadoc of the method are shown in Example 16-1.
The main navigation is driven by two attributes: viewId and webApp.

• webApp determines which web application to open.
• viewId determines the view activity within the target web application.

pageParametersList defines custom URL parameters that you can define.
The three parameters navTaskFlowId, navTaskParametersList, and navTaskLabel, provide support for loading a task flow on the target page, in addition to the default Main Tasks of the target page.

FndMethodParameters are placed as a future extension mechanism. The parameters that can be set through FndMethodParameters are: contextualAreaWidth and contextualAreaCollapsed. Example 16-2 shows how to set up these two parameters.
Note:

- The task flow to be opened in the target Work Area does not need to be pre-registered as a `defaultMain` or `dynamicMain` task for that page. The TaskFlow Id, parameters for it, and the label are all explicitly passed to the target page, and the target page does not perform any validation.

- The `webApp` value that is passed into the Navigate API is used to look up the host and port of the associated WorkArea or Dashboard from the Oracle Fusion Applications Functional Core (ASK) deployment tables. These tables are populated at deployment time through Functional Setup Manager tasks.

- Because the Navigate API is based on URL redirect, only string representations of the parameter values can be passed to the target page. It is not possible to pass an object as a parameter value. For example, a Java map that is set as a `customObject` on `FndMethodParameters` cannot be passed as a parameter (this is only supported in `openMainTask` API).

- When navigating to the current page with the `forcePageRefresh=true` option using this API, the ADF Controller state of the source page is not cleaned up.

16.1.2 How to Implement Navigation Across Web Applications

Navigating across web applications requires the `webApp` parameter in the menu meta data. For this parameter to work properly, the ApplicationDB and the AppMasterDB connections must be set. The `webApp` parameter must be specified for each parent node in the menu meta data. Child nodes inherit the value of this parameter from the parent node if not specified. The Navigate API is also modified to add the `webApp` parameter. Similarly, when using the `openSubTask` API, you must specify the `webApp` parameter correctly. There are no design or compile time checks that can catch an invalid value for the `webApp` parameter. It will throw null pointer exceptions at runtime only.

Note that you need to pass the `DEPLOYED_MODULE_NAME` in the ASK_DEPLOYED_MODULES table as the `webApp` parameter. The `DEPLOYED_MODULE_NAME`, by standard, should be the same as the context root of the application you are trying to open.

16.2 Warning of Pending Changes in the UI Shell

When there are pending changes in the UI Shell main area, and the user is leaving the page, or is refreshing the tab or task flow, or is closing the tab, the UI Shell will provide a modal confirmation dialog to the user. If confirmed, the operation will be allowed to proceed. Otherwise, the user will remain on the original page or tab.

Cases where pending changes are checked in the UI Shell main area include:

- TaskList links in regional area relaunch a task flow in the main area (with `reuseInstance=true` and either `forceRefresh=true` or with different parameters).

- Closing a tab by clicking the Close icon on the tab.
• Closing the currently focused tab by using closeMainTask.
• Relaunching a task flow by using openMainTask (with reuseInstance=true and either forceRefresh=true or with different parameters).
• Relaunching a task flow using navigate (navigating within the same web application and viewId).
• Opening a different work area or web application using navigate.

Search Panel and Warning of Pending Changes

Search is treated as a special case and no warning for pending changes is shown when a user enters some data in a query panel provided by the Oracle Application Development Framework. But, from the search results page, drilling down to a sub flow and making the flow dirty marks the sub flow as a candidate for warn about changes.

16.2.1 How to Implement Warning of Pending Changes

To implement warning of pending changes, developers need to follow these steps.

• All the main taskflows that render in MainArea should have the data-control-scope set to isolated.

```xml
<data-control-scope id="dc">
  <isolated/>
</data-control-scope>
```

• Make changes to the main area.

Note that inner taskflows and regions inside the main flow can have the data-control-scope set to shared.

Other than tasklist, if developers are using Data Control APIs, such as openMainTask, closeMainTask or navigate, to relaunch or close a flow in the MainArea, they should add the clientListener to post the changes. Adding the clientListener on a commandButton that is bound to closeMainTask to post pending changes before the currently focused tab/flow is closed is shown in Example 16-3.

When a command link/button invokes Data Control APIs programmatically, developers should add a client listener on the command link/button, as shown in Example 16-4.

In addition to adding the clientListener, when Data Control APIs are executed from within the MainArea, developers must add the methodAction shown in Example 16-5 to their main page fragment's pageDef whose taskflow is attached to the tab in MainArea.

The Data Control API shown in Example 16-5 lets you check for data dirty from within the child region and identify any pending changes in the child region.

After adding above API to the main page fragment's pageDef, developers should let the UI Shell know about that by sending the parameter "fndCheckDataDirty=true" in the parametersList or navTaskParametersList.

Example 16-3 Adding the clientListener on a commandButton

```af:commandButton actionListener="#{bindings.closeMainTask.execute}"
text="closeMainTask- with - CL"
```
Example 16-4   Adding a client listener when a command link/button invokes Data Control APIs programmatically

<af:commandButton text="Calling datacontrol api programmatically"
    binding="#{backingBeanScope.backing_Navigateviaprogrammatically.cb1}"
    id="cb1" action="go">
    <af:clientListener method="queueActionEventOnMainArea" type="action"/>
</af:commandButton>

Example 16-5   Adding the checkDataDirty methodAction

<methodAction id="checkDataDirty"
    InstanceName="FndUIShellController.dataProvider"
    DataControl="FndUIShellController" RequiresUpdateModel="true"
    Action="invokeMethod" MethodName="checkDataDirty"
    IsViewObjectMethod="false"
    ReturnName="FndUIShellController.methodResults.checkDataDirty_FndUIShellController_dataProvider_checkDataDirty_result"/>

16.2.2 How to Suppress Warning of Pending Changes

Developers can suppress warning of pending changes for a particular flow by sending the parameter "fndWarnChanges=false" to the parametersList or navTaskParametersList, as shown in Example 16-6.

Example 16-6   Suppressing warning of pending changes for a flow

<methodAction id="openMainTask" RequiresUpdateModel="true"
    Action="invokeMethod" MethodName="openMainTask"
    IsViewObjectMethod="false" DataControl="FndUIShellController"
    InstanceName="FndUIShellController.dataProvider"
    ReturnName="FndUIShellController.methodResults.openMainTask_FndUIShellController_dataProvider_openMainTask_result">
    <NamedData NDName="taskFlowId" NDValue="/WEB-INF/task-flow-definition.xml#task-flow-definition" NDType="java.lang.String"/>
    <NamedData NDName="keyList" NDType="java.lang.String"/>
    <NamedData NDName="parametersList" NDValue="fndWarnChanges=false" NDType="java.lang.String"/>
    <NamedData NDName="label" NDValue="Suppress Warn About Msg" NDType="java.lang.String"/>
    <NamedData NDName="reuseInstance" NDValue="true" NDType="java.lang.Boolean"/>
    <NamedData NDName="forceRefresh" NDValue="true" NDType="java.lang.Boolean"/>
    <NamedData NDName="loadDependentFlow" NDType="java.lang.Boolean"/>
    <NamedData NDName="methodParameters" NDType="oracle.apps.fnd.applicore.patterns.uishell.ui.bean.FndMethodParameters"/>
</methodAction>

16.3 Implementing the Oracle Fusion Home Page UI

The Oracle Fusion Home Page UI consists of a series of JSPX pages, each of which provides product- or role-specific content, that are visually tied together using a tabbed navigation interface.
16.3.1 Supported Behavior

The following key requirements are supported by the Oracle Fusion UI Shell:

- To conform to Oracle Fusion Applications modularity requirements, the Oracle Fusion Home provides a common entry point across multiple Java EE web applications. A given home page can be hosted on any one of these distinct Java EE web applications. A tab click on a given home page therefore must issue a request for another home page that may be hosted on a different web application. When the new page is displayed, its tab must be visually selected.

- Each home page should display content that includes navigation means to resources that may be hosted on a different web application. For example, a command navigation link on a home page may target a bounded task flow located on a different web application and designed to run on a Workarea or Dashboard page located on that web application.

- A given home page may include content from a bounded task flow (presumably displayed within an ADF region or portlet) that has command navigation links within its view activities. These links or other navigation means must work correctly on the home page. For example, on click, navigate to the correct Workarea page and launch the intended task flow.

- The Home icon in the Global Area will be disabled on all home pages.

- By default, clicking the Home icon will open the Welcome tab and, if a different tab is selected, clicking the Home icon will go back to the last tab that was selected for that session. If, however, the user navigates to any Oracle Fusion page through a direct URL navigation, such as an email that contains a link to a Workarea, the Home icon will again return to the Welcome tab. By default, the Welcome tab will be the first itemNode defined.

- A home page will display the same layout and content in the Global Area and for the page footer as other pages that extend the UI Shell page template. Unlike a Workarea page, however, the intervening content and its layout between the Global (top) and footer (bottom) sections of the page will be determined and provided by product teams.

16.3.2 How to Create a Home Page

To create a home page, follow these basic steps.

1. Create a JSPX page that extends UIShell.jspx.

2. From the JSPX page, select the <af:pageTemplate> tag. On the property inspector, set the isHomePage attribute to true. Note that when isHomePage is set to true, the Regional, Local and Contextual Areas of the UI Shell will not be rendered.

3. Add content to the HomePageContent facet, following the ADF Layout Basics guideline for laying out the components.
4. Drop the JSPX to adfc-config.xml.

5. Repeat Steps 1 through 4 for all home page JSPX pages.

6. Create the Home Page menu. See Working with the Global Menu Model.

7. When running a home page JSPX, the page should look similar to Figure 16-3.

**Figure 16-3  Home Page Example**

See Global Area Standard Icons for a description of the icons.

### 16.3.3 Getting the URL

There are two APIs located in the UIShellContext class that you can use to return a URL without actually performing the navigation to the URL:

- **getURL**: getURL is the same as a navigate call, but it returns the URL as a string instead of doing the navigation. See Example 16-7.

- **getURLForCurrentTask**: getURLForCurrentTask, shown in Example 16-8, is good for getting the URL of the main area flow in focus so you could send it in an email to someone else. When clicked, it would open the correct workarea (.jspx page) and the correct flow with the correct parameters. If you did have other dynamic tabs open when you called getURLForCurrentTask, it would not open those when clicked.

**Example 16-7  Example Use of getURL**

```java
public java.lang.String getURL(     java.lang.String    viewId,
    java.lang.String    webApp,
    java.lang.String    pageParametersList,
    java.lang.String    navTaskFlowId,
    java.lang.String    navTaskKeyList,
    java.lang.String    navTaskParametersList,
    java.lang.String    navTaskLabel,
    FndMethodParameters methodParameters)
    throws java.io.IOException
```

**Example 16-8  Example Use of getURLForCurrentTask**

```java
public java.lang.String getURLForCurrentTask( )
    throws java.io.IOException
```

### 16.4 Using the Single Object Context Workarea

The Single Object Context Workarea is a facet that defines an area between the Global Area and the Main and Regional areas into which developers can add what they want. If the facet is empty, the area does not appear. You also can create a view scope parameter so other flows on the page can get the context. See Figure 16-4.
This area is useful for creating Single Object Workareas, which are particularly useful in tabbed page mode. A Single Object Workarea is a page that is devoted to one object so the information of that one object can be put above the Workarea.

Single Object Workareas provide a context for addressing the tasks and processes for the business process of a single complex object instance at a time. Usually, single object workareas will use multiple defaultMain flows. Each flow can be about a different aspect of the same object. For instance, you could have one tab showing recent activity, and another tab showing payments.

### 16.4.1 Implementation Notes

The SingleObjectContextArea facet has been added to the UI Shell template for Context Area task flow.

A managed bean entry, shown in Example 16-9, has been added to adfc-config.xml for viewScope Hashmap.

Support for URL parameters defined in the view activity for the page (JSPX) in adfc-\config is to be set in the fndPageParams Hashmap defined in viewScope. This Hashmap is also passed onto the pageFlowScope of the task flows in the context area, regional area and main area.

This is accomplished by enabling the Bookmarkable property on the view activity for the page and specifying the URL parameter names with the value set to be added to the fndPageParams Hashmap defined in viewScope. See the “How to Create a Bookmarkable View Activity” section in the Developing Fusion Web Applications with Oracle Application Development Framework.

Product team task flows running in the Single Object Context Workarea must be designed to take in the appropriate context values as input parameters. The pageFlowScope values must be passed to the appropriate input parameters through the parametersList for the item node in the menu or the openMainTask API.

Product teams can cause URL navigation (using the navigate data control method provided by UI Shell by passing required parameters in the pageParametersList) for changing context or updating context information. This will cause the entire page to be refreshed based on new context parameter values.

Product teams can also check for the input parameter values (the context) within the context area task flow. If invalid, a modal dialog can be launched for the end user to
choose a context (rendered as links based on the navigate data control method provided by UI Shell) before proceeding.

**Example 16-9   Managed Bean viewScope Hashmap Entry in adfc-config.xml**

```xml
<managed-bean>
    <managed-bean-name>fndPageParams</managed-bean-name>
    <managed-bean-class>java.util.HashMap</managed-bean-class>
    <managed-bean-scope>view</managed-bean-scope>
</managed-bean>
```

16.4.1.1 Developer Implementation

Enable the Bookmarkable property on the view activity for the single object context page and specify the URL parameter names with the corresponding fndPageParams viewScope Hashmap key where the value should be stored. See the "How to Create a Bookmarkable View Activity" section in the Developing Fusion Web Applications with Oracle Application Development Framework.

**Example 16-10** shows a sample entry in adfc-config.xml for such a view activity.

Product teams must create a bounded task flow for the Context Area that appears at the top of the page. This task flow must accept the context values as input parameters. This task flow is dropped as a static region onto the Context Area facet in the JSPX page based on the UI Shell template. In the task flow binding, set the input parameter values to the corresponding key in fndPageParms viewScope Hashmap.

**Example 16-11** shows a sample entry in the page definition for the JSPX file for passing the viewScope values into the Context Area task flow as input parameter.

The UI Shell code passes this viewScope hashmap onto the pageFlowScope of the main area and regional area task flows. Product team task flows, which are initialized in the regional area and main area, can access this in pageFlowScope (of a main area or regional area container task flow owned by Applications Core) for the appropriate input parameters through the menu model and openMainTask API.

**Example 16-12** shows a sample entry for a child item node for defaultMain task in the menu.xml file to pass the appropriate deptno in the single object context to a task flow that is initialized based on this input value.

Product teams can cause URL navigation (through the navigate data control method provided by UI Shell) for changing context or after updating context information. This will cause the entire page to be refreshed based on new context parameter values.

Product teams can also check for the input parameter values within the pageFlowScope of the context area task flow. If invalid, a modal dialog can be launched for the end user to choose a context (by providing links based on the navigate data control method provided by the UI Shell by passing required parameters in the pageParametersList) before proceeding. This can be achieved by defining an invoke action executable in the page def for the context area page fragment which checks for existence of the pageFlowScope value and programmatically shows a modal dialog for the user to choose the context. The other alternative is to use this approach to cause navigation to a completely different page where the user can select the context.

Also, it is important to note that all the bounded task flows that will be initialized in the Single Object Context Workarea need to check for input parameter values and show/query data only if the input parameter values are passed. Basically, if the input parameter values are empty, the task flows handle that, such as in a router activity, and avoid showing any transaction data to the end user.
16.5 Implementing the Third-Party Component Area

The Third-Party Component Area, shown in Figure 16-5, is a facet in the UI Shell template for showing content just above the Single Context Workarea in the global area. Although originally designed to contain a Call Telephony Interface (CTI) showing incoming calls in a call center operation, the facet can contain anything.

**Figure 16-5  Third Party Component Area Facet Containing A CTI**

16.5.1 How to Implement the ThirdPartyComponentArea Facet Developer

The ThirdPartyComponentArea facet should be added within a flexible layout component that will allow you to control the amount of screen real estate that the content added to the facet can consume.

- A facet reference is under the global area content with its width set to 100 percent.
- Add content to this facet.
• You should create a bounded task flow with page fragment for the content and drop it into this facet as a static region. This supports personalization through web composer since the content is within a page template.

• The height property of the panelStretchLayout in the UI Shell template defaults to 33 px. To view or enable the third party component area (see the dark blue area at the top of Figure 14-1), specify the globalAreaHeight property as auto in the pageTemplate property Inspector.

When the height of the top facet is specified as a CSS length or as auto, this facet will no longer be stretched and instead will consume the initial offsetHeight given to its children by the browser. It is important to note that in an average page, a switch to a layout using automatic heights exhibited a 5- to 10-percent degradation in initial layout speed. Also, an automatic height will cause the facet child to not be stretched both vertically and horizontally.

• You can set the height of the root layout component within the facet to auto to ensure auto-resizing of contents within the facet. Consider a showDetailHeader that is the root element within the facet (in the default view activity for the bounded task flow dropped in as a region). If the height property for this showDetailHeader component is set to auto, collapsing the header would resize the contents and open more real estate in the screen for showing other components in the page.

16.6 Developing an Activity Guide Client Application with the UI Shell

The Oracle UI Shell can be used to develop an Activity Guide client application.

Before you begin:

Copy the file oracle.bpm.activityguide-ui_11.1.1.jar to a local directory from the following location in the bpm-jdev-extension.zip file:

jdev_install/jdeveloper/soa/modules/oracle.bpm.activityguide-ui_11.1.1.jar

16.6.1 How to Develop an Activity Guide Client Application Using Oracle UI Shell

To develop an activity guide client application using Oracle UI Shell, follow these steps:

1. Create a new application. In the Application Package Prefix field, enter oracle.ag.
When prompted to create a new project, click **Cancel**.

2. In the new application, create a new project using the Web Project template by right-clicking the application and selecting **New Project > Project > Web Project**. Select **Servlet 2.5/JSP 2.1 (Java EE 1.5)**. Click through the wizard, accepting all default values.

3. Add the **ADF Faces** and **ADF Page Flow** technologies to the client application project. Right-click the project and select **Project Properties > Technology Scope**. Shuttle **ADF Faces** and **ADF Page Flow** from the Available Technologies list to the Selected Technologies list.

Make sure the following technologies are also selected: HTML, Java, JSP and JSF and Servlets.
Figure 16-7 Adding the ADF Faces and ADF Page Flow Technologies to the Client Application

4. Right-click the client application and select Project Properties > JSP Tag Libraries. Select the Distributed Libraries folder and click Add. In the Choose Tag Libraries window, select the tag library Applications Core (ViewController) 11.1.1.0.0 and click OK.

Figure 16-8 Select the Tag Library Applications Core (ViewController) 11.1.1.00

5. In the Project Properties window, select Libraries and Classpath and click the Add Library button. From the window that is displayed, select the Applications Core library and click OK.
6. In the Project Properties window, click the **Add JAR/Directory** button and browse for the file `oracle.bpm.activityguide-ui_11.1.1.jar`. The file is located under `jdev_install/jdeveloper/soa/modules/oracle.bpm.activityguide-ui_11.1.1.jar`. Click **Select** to add the JAR to the project classpath.

7. Add Activity Guide runtime libraries or JAR files to the classpath. Use either shared libraries or JAR files; do not use both.

   a. Using shared libraries for Activity Guide runtime JAR files

      Add the shared library references `oracle.soa.bpmel` and `oracle.soa.workflow.wc` to the `weblogic-application.xml` file.

      ```xml
      <library-ref>
        <library-name>oracle.soa.bpmel</library-name>
      </library-ref>
      <library-ref>
        <library-name>oracle.soa.workflow.wc</library-name>
      </library-ref>
      
   b. Using JAR files for Activity Guide runtime

      Add the following JAR files to the classpath:

      ```
      FMW_home/AS11gR1SOA/soa/modules/oracle.soa.workflow_11.1.1/bpm-services.jar
      FMW_home/AS11gR1SOA/soa/modules/oracle.soa.bpmel_11.1.1/orabpel-common.jar
      FMW_home/AS11gR1SOA/soa/modules/oracle.soa.bpmel_11.1.1/orabpel.jar
      FMW_home/AS11gR1SOA/soa/modules/oracle.soa.fabric_11.1.1/bpm-infra.jar
      FMW_home/AS11gR1SOA/soa/modules/oracle.soa.fabric_11.1.1/fabric-runtime.jar
      FMW_home/oracle_common/modules/oracle.webservices_11.1.1/wsclient.jar
      FMW_home/oracle_common/modules/oracle.xdk_11.1.1/xml.jar
      ```

8. Create a file called `Config.jar` using the `wf_client_config.xml` file and add it to the application classpath. The `wf_client_config.xml` file should include the host name and port of the WLS instance running the Activity Guide instances.
9. Create a JSF page. Right-click the application name and click **New**. In the New Gallery, select **JSF > JSF Page** and click **OK**. Use **UIShell** as a page template and select the checkbox **Create as XML Document (*.jspx)**.

If a dialog box displays the message "Confirm Add Form Element," click **No**.

10. Create Application menu metadata. See **Implementing Application Menu Security**.

### 16.6.2 How to add a task flow to the Oracle UI Shell client application

Follow these steps to add a task flow to the Oracle UI Shell client application:

1. Open the menu metadata menu XML file created in "How to Develop an Activity Guide Client Application Using Oracle UI Shell".

2. Right-click itemNode_<JSF page name> and select **Insert inside itemNode_<JSF page name> > itemNode**.

3. In the Common Properties window, browse for the name of the JSF page and enter a unique Id for the itemNode property using the standard format <pageID>_<taskFlowName>. For example: ItemNode_MainArea_TaskFlow.

4. Click the **Browse** button to the right of the focusViewId field. In the Edit Property window that is displayed, select the focusViewId of the page under which you are registering the task flow. Click **OK**.

5. In the Property Inspector, select the Applications tab and enter a label for the itemNode such as **Main Taskflow**.

6. From the Project Navigator, select the menu metadata XML file. In the Structure view, select the task flow itemNode - Main Taskflow. In the Property Inspector, enter the following values under the Advanced section:

   - **Task Type**: defaultMain
   - **Task Flow ID**: Click the **Browse** button to display the Select Task Flow ID window and select the location of the task flow definition: `/WEB-INF/oracle/bpel/activityguide/ui/taskflows/ag-humantask-task-flow.xml#ag-humantask-task-flow`. This is the Id of the main area task flow.
   - **Disclosed**: Optionally, set this value to **false**. This property enables opening a new tab when clicking the relevant task link at run time. For tabs, the first defaultMain with disclosed=true is the one that will be in focus.

7. Repeat Steps 1 through 6 for the regional task flow, naming the label and Id accordingly. Provide the following values for the regional task flow in step 6:

   - **Task Type**: defaultRegional
   - **Task Flow ID**: Click the **Browse** button to display the Select Task Flow ID window and select the location of the task flow definition: '/WEB-INF/oracle/bpel/activityguide/ui/taskflows/ag-tasktree-task-flow.xml#ag-tasktree-task-flow'. This is the Id of the regional area task flow.
   - **Disclosed**: Set this value to **true** for the regional task flow. This property enables opening a new tab when clicking the relevant task link at run time.
8. Create a file called `activityguide.properties`. If you choose another name then you must provide its value to the ag-tasktree-task-flow using the parameter `AGPropsBeanName`.

If using identity propagation to secure the Activity Guide, the properties `WorkflowAdminUser` and `WorkflowAdminPassword` are not required.

You can customize Activity Guides behavior by configuring their properties. To configure these properties you must edit the Activity Guide properties file.

Table 16-3 shows the properties you can specify in this file.

Example 16-13 shows a typical Activity Guide properties file:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
</table>
| ServerConnectionMode    | Specifies the mode for the transmission of data.                           | • SOAP  
                          |                                                              | • REMOTE |
| WorklistHttpURL         | Only required when using digital signatures. Specifies the URL to access the  | http://host:port/integration/                        |
                          | Oracle BPM Worklist application.                                            | worklistapp                                          |
| SelectionFilter         | Specifies the filter used to filter the processes in an activity guide.      | • MY  
                          |                                                              | • PREVIOUS  
                          |                                                              | • REPORTEES  
                          |                                                              | • ADMIN |
| AGDefinitionFilter      | Specifies the definition ID used to filter the process in an activity guide. | `activity guide definition ID`                      |
| AGInstanceOrdering      | Specifies the order used to display the processes in the activity guide.     | • `column_name`:ASC  
                          | For example: CREATION_DATE:ASC                                              | • `column_name`:DESC  
                          |                                                              | Default value: ASC |
| AGInstanceID            | Specifies the instance ID used to display the activity guide tree.           | `activity guide instance ID`                        |
|                         | For example: 10001                                                          |                                                      |
| CustomPredicate1        | Specifies an additional predicate to filter the list of processes in an activity guide. For example: CREATOR, EQ, jstein | `column name, operator, value`   |
| CustomPredicate2        | Specifies a different additional predicate to filter the list of processes in an activity guide. This predicate is used with CustomPredicate1 | `column name, operator, value`   |
| ShowAllAGTreeNodesProperties | Specifies if the activity guide shows a section at the top that describes the properties of activity guides, milestones and tasks. | • true  
                          |                                                              | • false  
                          |                                                              | Default value: true |
| ShowRefreshButton       | Specifies if the regional area displays a refresh button.                    | • true  
                          |                                                              | • false  
                          |                                                              | Default value: false |
### Table 16-3 (Cont.) Activity Guide Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGTasksPopupTaskFlowID</td>
<td>Specifies the content to display in the task pop-up.</td>
<td>fully qualified TaskFlow ID</td>
</tr>
<tr>
<td>HideAGTreeRootNode</td>
<td>Hides the Guided Business Process title on the Activity Guide Tree root node.</td>
<td>true, false</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
<td></td>
</tr>
<tr>
<td>ShowCustomBlockedIcon</td>
<td>Specifies if the Guided Business Process shows the custom task blocked icon.</td>
<td>true, false</td>
</tr>
<tr>
<td></td>
<td>Default value: false</td>
<td></td>
</tr>
</tbody>
</table>

9. In the page definition of Oracle UI Shell JSF fragment page, navigate to `pageTemplateBinding` and set the Refresh property to `ifNeeded`.

10. Open the file `adfc-config.xml`.

11. Edit the file `adfc-config.xml` to include the location of the `activity.properties` file. This should be the absolute path to the `activityguide.properties` file.

   An example `adfc-config.xml` is shown in **Example 16-14**.

12. To enable a task flow popup with summary information, add the property `AGTasksPopupTaskFlowID` to the `activityguide.properties` file.

   Use this parameter to display a task flow summary in dynamic regions. Enter the relevant task flow Id. If this parameter is not set, the value of `OutputText` is shown as the default task summary.

13. Create a Workflow Service client configuration file. An example is shown in **Example 16-15**.

14. Deploy the Activity Guide client application with Oracle UI Shell as described in Deploying an Activity Guide Client Application with Oracle UI Shell to Integrated or in Deploying an Activity Guide Client Application with Oracle UI Shell to Standalone.

**Example 16-13 An Activity Guide Properties File**

```xml
#ActivityGuide Properties
ServerConnectionMode=SOAP
SelectionFilter=MY
ShowRefreshButton=true
#ShowAllAGTreeNodesProperties=true
## Sample value for AGDefinitionFilter: default/BPMAGPrj2!2.0*31fcd931-6263-4b58-97cf-6fb084addabc
#AGDefinitionFilter=
#AGDefinitionFilter=
#AGInstanceID=110003
#AGInstanceOrdering=CIKEY:DESC
#CustomPredicate1=STATE,EQ,OPEN
#CustomPredicate2=STATUS,EQ,In Progress
## Example Value for AGTasksPopupTaskflowID is /WEB-INF/ag-popup-task-flow.xml#ag-p
#AGTasksPopupTaskflowID=
#ShowCustomBlockedIcon=true
#HideAGTreeRootNode=false
```
## WorklistHttpURL is required only for digital signatures

```
#WorklistHttpURL=http://host:port/integration/worklistapp
```

### Example 16-14  `adfc-config.xml` File with Reference to `activityguide.properties` File

```xml
<managed-bean>
  <managed-bean-name>agProps</managed-bean-name>
  <managed-bean-class>
    oracle.bpel.activityguide.ui.beans.model.AGProperties
  </managed-bean-class>
  <managed-bean-scope>request</managed-bean-scope>
  <managed-property>
    <property-name>absAgPropsFileName</property-name>
    <property-class>java.lang.String</property-class>
    <value> <!---absolute path on your machine should be given here-->/activityguide.properties
      <!-- For example:
        Windows: C:\AG\activityguide.properties
        Linux: /scratch/<user>/AG/activityguide.properties
      -->
    </value>
  </managed-property>
</managed-bean>
```

### Example 16-15  Workflow Services Client Configuration File

```xml
<workflowServicesClientConfiguration xmlns="http://xmlns.oracle.com/bpel/services/client">
  <server default="true" name="default">
    <localClient>
      <participateInClientTransaction>false</participateInClientTransaction>
    </localClient>
    <remoteClient>
      t3://host:port
      <initialContextFactory>weblogic.jndi.WLInitialContextFactory</initialContextFactory>
      <participateInClientTransaction>false</participateInClientTransaction>
    </remoteClient>
    <soapClient>
      http://host:port
      <identityPropagation mode="dynamic" type="saml">
        <policy-references>
          <policy-reference enabled="true" category="security" uri="oracle/wss10_saml_token_client_policy"/>
        </policy-references>
      </identityPropagation>
    </soapClient>
  </server>
</workflowServicesClientConfiguration>
```

### 16.6.3 How to Secure the Activity Guide Oracle UI Shell Client Application

Securing the Activity Guide client application ensures that only users with proper credentials can complete the tasks outlined in the Activity Guide. Security features include authentication, authorization, realm verification and policy enforcement.

Follow the instructions for securing a Web application as described in the Enabling ADF Security in an Oracle Fusion Web Application section of the *Oracle Fusion Middleware Developing Fusion Web Applications with Oracle Application Development Framework*.
You can deploy an Activity Guide client application with Oracle UI Shell directly from JDeveloper or to the standalone Oracle WebLogic Server.

### 16.6.3.1 Deploy an Activity Guide Client Application with Oracle UI Shell to Integrated Oracle WebLogic Server

From the Application Navigator, right-click the JSF page created in How to Develop an Activity Guide Client Application Using Oracle UI Shell and select Run.

### 16.6.3.2 Deploy an Activity Guide Client Application with Oracle UI Shell to Standalone Oracle WebLogic Server

Follow these steps to deploy an Activity Guide client application with Oracle UI Shell to a standalone Oracle WebLogic Server:

1. Create a connection to the standalone Oracle WebLogic Server.
2. From the Application Navigator, right-click the project created in step 2 of How to Develop an Activity Guide Client Application Using Oracle UI Shell.
3. Select Project Properties.
   - From the Project Properties dialog, select Deployment.
4. Create a new WAR deployment profile.
5. Right-click the project and select Deploy. Deploy the project to the standalone Oracle WebLogic Server connection created in step 1.
6. Launch the client page from a browser.

### 16.7 Implementing Touch Device Support in the UI Shell

The Global Area display on a tablet device is the same as the standard Global Area design except these menu options and display area are not available:

- **Administration** menu options
- **Personalization > Customize** this page menu option

The items that are not available on a tablet device take up too much space to be used comfortably on a small screen.

The page template will also collapse the left-side Regional area by default to save space.

Also see the Creating Web Applications for Touch Devices Using ADF Faces section in the Oracle Fusion Middleware Developing Web User Interfaces with Oracle ADF Face guide.

**How to use flow mode on a touch screen device**

Implementing flow mode follows the Best Practices When Using ADF Faces Components in a Mobile Browser section in the Oracle Fusion Middleware Developing Web User Interfaces with Oracle ADF Face guide.

To use flow mode on a touch screen device, but making sure that nothing is changed when displaying pages on a desktop system, add this `context-param` to the `web.xml` file of the J2EE application:

```xml
<context-param>
  <description>
    This parameter controls the default value for component geometry on the page.
  </description>
</context-param>
```
Supported values are:
  legacy - component attributes use the default values as specified for
  the attributes
  auto - component attributes use the correct default value given the
  value of their parent component. For example, with this setting, the
  panelStretchLayout will use "auto" as the default value for its
  "dimensionsFrom" attribute instead of "parent".
</description>
<param-name>oracle.adf.view.rich.geometry.DEFAULT_DIMENSIONS</param-name>
<param-value>#{requestContext.agent.capabilities['touchScreen'] eq 'none' ? 'legacy' : 'auto'}</param-value>
</context-param>

In this case, the dimensionsFrom attribute is set to auto only when running on a
touch screen device. But when running on a desktop system, the context-value is
set to legacy and the dimensionsFrom attribute will use the old default value,
which is parent.

The UI Shell sets the dimensionsFrom attribute to children on the root panel
component on a page when running on a tablet device. Setting the dimensionsFrom
attribute to auto, and placing the context-param in the web.xml file, will allow all
other components to use the dimensionsFrom attribute.

16.8 Implementing the Translation Editor

The Translation Editor is a standard UI component that allows translated data to be
edited in all languages, regardless of the session language. It is accessible in Table
layout. It also:

- Allows all translated data stored in multi-lingual (TL) tables to be edited on the
  same setup UI as untranslated data for the same object.
- Automatically synchronizes any updates required by source_lang defaulting.
- Does not require the user to change session language, even temporarily.

The Translation Editor is implemented as a popup window, launched at the row level.
The Edit Translations popup has a row for each language that is installed and
implemented by the Translation Editor. When the user highlights an editable row and
clicks the Translations Editor button and the popup displays, the user can enter, for
instance, a description in German in the row marked German. This then places the
German text in the correct field in a database.

This example shows that the ABSENCE_STATUS row is selected:
Click the Translation Editor button to display the Edit Translations dialog:

A user can enter other language descriptions in any other available language row without having to close the Edit Translations dialog and restarting it.

Assume the user’s job is to enter data in multiple languages: Korean, German and English. When the Edit Translations dialog opens, the user would see Korean, German, and English rows.

When the system first starts, there may be no data. When data is added, it would be in a single language. That is the source language. So if a Korean user first adds the data, when any user views the data, it would be in Korean, even if the user were an English or German user. Then when the translation editor opens, it would have Korean data in all rows, and the Source Language in the right-hand column for each row would say Korean in all rows. If a German user then opens the translation editor and changes the German row, the Source Language for that row will change to German.

Note: If the source language value is updated, all values that show the same source language also will update.

Note: A user can enter a translation in the wrong field; such as entering a German translation in the French field.

Limiting languages shown: This is to help someone that is translating many UIs for a certain language to limit the popup to only show the language on which he or she is currently working. This control is accomplished by setting the profile option FND_TRANS_EDITOR_LANGS to the languages that you want to see. This profile can be set at Site or User level.

To access the profile option, click Setup and Maintenance > Manage Administrator Profile Option Values > then search for FND_TRANS_EDITOR_LANGS.
16.8.1 How to Implement the Translation Editor

**Important:** This section assumes that developers will have completed all coding necessary to support the OA Base Class Translation Model. See OADEE58122 “Using Oracle Fusion Middleware Extensions for Oracle Applications Base Classes.”

To implement the Translation Editor, follow these steps:

1. Define a view object for the translatable table (TL). For more information, see the Defining SQL Queries Using View Objects section in the *Oracle Fusion Middleware Developing Fusion Web Applications with Oracle Application Development Framework.*

   If the Multi-language support (MLS) coding standard has been followed (see Understanding Multi-Language Support Features), every TL should have an entity object that is linked to an entity object based on the view link (VL) view, but only the VL-based entity object has a view object. To support the Translation Editor, a view object also must be added for the TL entity object. Define a new view object based on the TL entity object. The view object should be the default and should include all attributes in the TL entity object. No other changes are needed. This view object must be added to the Application Module as a child of the VL view object. The translated attributes in this new view object will inherit custom properties fnd:OA_TRANSLATABLE from the TL entity object that should have been set previously.

2. On the new TL view object, define the custom property at the view object level to `fnd:OA_TRANSLATION_TABLE = true`, as shown here.
The translated attributes in this new view object will inherit custom properties fnd:OA_TRANSLATABLE from the TL entity object that should have been set previously.

3. Add a view link for the view object. This new TL view object must be a child of the base VL view object. A view link must be defined between the VL view object and the new TL view object.

4. Copy validators from VL entity object to TL entity object. Using the MLS coding standard, only the VL entity object will have any validation; the TL entity object has no validation, it relies on validation in the VL entity object before values are copied to the TL entity object by the OA base classes. This will not be sufficient though if translated values can be updated directly. Any validators on the VL entity object that reference translated attributes must be copied to the TL entity object.

5. Copy control hints from VL entity object to TL entity object. If you did not set the control hints to your TL entity object, add control hints to the TL entity object, for prompt, display length, widget, and so on. These control hints will be used when rendering the Translation Editor. Generally these will be the same control hints as the attributes on the VL entity object, but under some circumstances might be different (for example, display length may be shorter.)

6. Set the display name for the Language and SourceLang attributes. Note that the TL view object needs to join to LanguagePEO so that you get the correct display name for the Language and SourceLang attributes.

- To do that, you will need to create two entity object associations. One is from your Language attribute to the LanguagePEO.LanguageCode. The other one is from your SourceLang attribute to the LanguagePEO.LanguageCode. For example, the two entity object associations created for this test case are LookupTypeTranslationVOLanguageToLanguagePEO and lookupTypeTranslationVOSourceLangToLanguagePEO.

Look for LanguagePEO as shown here and add it to the TL view object as an entity object.
You will need to add it twice. Once for the Language attribute that would use the association `LookupTypeTranslationVOLanguageToLanguagePEO` and once for the SourceLang attribute that would use `LookupTypeTranslationVOSourceLangToLanguagePEO`. The two `LanguagePEO` instances should be named `LanguagePEOForLanguage` and `LanguagePEOForSourceLang`.

Add the `LanguageDisplayName` and `SourceDisplayName` attributes into your translation view object.
Figure 16-16 Adding Attributes to Translation View Object

In the translation popup window, you should show the language description instead of the language code, such as American instead of US.

Note:
When adding these attributes to the entity object, you must give them the Aliases LanguageDisplayName and SourceLangDisplayName.

Figure 16-17 Required Aliases for Attributes Added to Entity Object

The LanguageDisplayName and SourceLangDisplayName attributes now should be included in the TL view object.
Figure 16-18  Required Attributes Added to TL View Object

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Alias Name</th>
<th>Entity Usage</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>LookupType</td>
<td>String</td>
<td>LookupType</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>String</td>
<td>LANGUAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SourceLang</td>
<td>String</td>
<td>SOURCE_LANG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meaning</td>
<td>String</td>
<td>MEANING</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Description</td>
<td>String</td>
<td>DESCRIPTION</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LanguageDisplayName</td>
<td>String</td>
<td>DESCRIPTION2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LanguageCode</td>
<td>String</td>
<td>LANGUAGE_CODE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SourceLangDisplayName</td>
<td>String</td>
<td>DESCRIPTION2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LanguageCode</td>
<td>String</td>
<td>LANGUAGE_CODE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note:

Set the Query Column Alias of the LanguageDisplayName attribute to LANGUAGE_DISPLAY_NAME. The Translation Editor uses the Language Display Name to set the order when displaying the translation data in the popup. You do not need to set the Query Column Alias for SourceLangDisplayName.

Figure 16-19  Setting the Query Column Alias of the LanguageDisplayName Attribute

The TL model should now be complete. The next step is to add the Translation Editor to the UI.

16.8.2 How to Add the Translation Editor to Existing Application Tables During Design Time

This section describes how to add the Translation Editor to an Application Table on an existing JSF page. If you are creating a new Application Table, skip to How to Add the Translation Editor to New Application Tables During Design Time.
You will use a design time wizard to add the Translation Editor to a UI.

For example, if you drag a translatable table (TL) view object instance from the dataControl to an existing ApplicationTable > additionalToolbarButton > af:toolbar or ApplicationTable > f:facet-popup > panelGroupLayout, you can pick the Translation Editor under applications. You can drop the Translation Editor on either facet; the result will be the same.

Note that you cannot drag and drop directly onto an ApplicationTable.

This shows an example drag-and-drop of a TL view object onto an ApplicationTable > additionToolbarButton-> af:toolbar.

**Figure 16-20 Example Drag-and-Drop of A TL View Object**

You also can drag and drop your TL view object onto ApplicationTable > f:facet-popup > panelGroupLayout.

After the drag-and-drop, the Create Translation Popup wizard will open. The wizard will include input for a list of all the columns in the TL view object and you can remove the ones you do not want. The list will show all the attributes in the TL view object, except for Who columns, LANGUAGE, SOURCE_LANG and the Primary Key.

You can remove the translation attributes you do not want to display. You also can pick input text field or text region as the type. All other display properties (such as prompts) will be defaulted from control hints on the TL view object.

**Figure 16-21 Create Translation Popup Wizard**

When you click **Continue**, designer will add the popup code into your JSFF file. You should not edit anything in the code that is added to your page.
16.8.3 How to Add the Translation Editor to New Application Tables During Design Time

The Application Table Configure Table Patterns wizard has an option for users to choose whether to have the translation editor ability. The View Instances that will be available to enable for the Translation Editor will be those view objects marked with the fnd:OA_TRANSLATION_TABLE custom property (see Model layer changes below). If there are no such view objects, the Translation Editor option will be disabled.

16.8.4 Handle Multi-Language Data Entry

The Translation Editor allows translated data to be edited in all languages, regardless of session language.

- Allow all translated data stored in multi-lingual (TL) tables to be edited on the same setup UI as untranslated data for the same object
- Automatically synchronizes any updates required by source_lang defaulting
- Does not require the user to change session language, even temporarily
- Accessible in either Form or Table layout
- Is a standard component that require minimal hand-coding by consuming developers
- Properly handles validation of translated values

Runtime Design

The Translation Editor is implemented as a popup window, launched at the row level by a button. The button and popup window will appear as shown in these two images:

Figure 16-22 Example of the Translation Editor in a Table
Implementing the Translation Editor

The translation button will be in the primary button region of Application Table. If there is only 1 language active, the icon will not be visible.

The popup window will show a table of translated data for a single data row, with 1 row per active language (translated data for disabled languages will not be displayed, even if present). The columns of the table will include language and source language (both read-only), and 1 column for each translated attribute. Translated attributes may be editable or read-only. Column prompts and display properties (such as display size) should match the prompt and display properties of the base attribute. (NOTE: TBD how to do this.)

When a value is updated:

- The row will be marked at "translated," meaning the language and source language will be set to the same value.
- Any other languages with the same source language as the updated row will be re-defaulted.

If the window is closed:

- If closed with OK, values will be written back to be committed (or not) with the main transaction. If any translated values for the session language have been updated, they will be updated in the main UI.
- If closed with Cancel, all changes in the popup are discarded without affecting anything in the main transaction.

When the entire transaction is committed, any updates performed in the Translation Editor will be validated and committed at the same time. If any validation errors occur in translated data, they will be reported along with all other validation errors. The user will need to re-open the popup window to correct the error. If the error is associated with an attribute, that attribute will have a "red box" indicator in the popup, only on the language row in error.

**Ability to limit languages shown:** The languages that are shown can be limited. This is to help someone that is translating many UIs for a certain language to limit the
popup to only show the language on which he or she is currently working. This control is accomplished by setting the profile option FND_TRANS_EDITOR_LANGS to the languages that will be shown. This profile can be set at SITE or USER level. If null, all active languages are shown. The values (US, KO, F) can be found in the Manage Languages setup UI.

*Figure 16-24 Setting the FND_TRANS_EDITOR_LANGS Profile*

Launching the Translation Editor from a form is shown in this example.

*Figure 16-25 Launching the Translation Editor from a Form*

When you click the Translation Editor button, the same popup described for the Application Table case will show.

*Figure 16-26 Edit Translations Popup*
If your form has multiple regions that involve more than one TL ViewObject, then we recommended that you drag-and-drop the TL ViewObject onto the desired location in your form. For example, in the below case, the Translation Editor button is placed at the end of the form.

**Figure 16-27 Edit Profile Category**

<table>
<thead>
<tr>
<th>Manage Profile Category</th>
<th>Edit Profile Category : FND/forms/UI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Category Code</strong></td>
<td><strong>FND/forms/UI</strong></td>
</tr>
<tr>
<td><strong>Category Name</strong></td>
<td><strong>Forms UI</strong></td>
</tr>
<tr>
<td>Application</td>
<td><strong>Oracle Middleware Extensions for Applications</strong></td>
</tr>
<tr>
<td><strong>Module</strong></td>
<td><strong>Application Common</strong></td>
</tr>
</tbody>
</table>

### 16.8.4.1 Implement Multi-Language Data Entry

**Prerequisite**

Before starting this, developers will have completed all coding necessary to support the OA Base Class Translation Model. See Understanding Multi-Language Support Features for setting up translation tables.

**Prepare the TL EO and Entity Associations**

1. Copy validators from VL EO to TL EO. In accordance with the MLS coding standard, only the VL EO will have any validation; the TL EO has no validation, it relies on validation in the VL EO before values are copied to the TL EO by the OA base classes. This will not be sufficient if translated values can be updated directly. Any validators on the VL EO that reference translated attributes must be copied to the TL EO.

2. Copy control hints from VL EO to TL EO. If you did not set the control hints to your TL EO, add control hints to the TL EO for prompt, display length, widget, and so on. These control hints will be used when rendering the Translation Editor. Generally, these will be the same control hints as the attributes on the VL EO, but under some circumstances might be different (for example display length may be shorter.)

3. Create Entity Associations for Language attributes. Create two EO associations from the TL EO to LanguagePEO: one from your Language attribute to the LanguagePEO.LanguageCode, and one from the SourceLang attribute to the LanguagePEO.LanguageCode. Both associations are 0,1 .. *, everything else can be left as default. For example, the two EO associations created for this test case are LookupTypeTranslationVOLanguageToLanguagePEO and LookupTypeTranslationVOSourceLangToLanguagePEO.
Create the TL VO and View Link

1. Define a VO for the TL table. If the MLS coding standard has been followed, every TL table should have an EO, linked to an EO based on the VL view, but only the VL-based EO has a VO. To support the Translation Editor, a VO must be added for the TL EO as well. Define a new VO based on the TL EO. The VO should be the default, and include all attributes in the TL EO. No other changes are needed. The translated attributes in this new VO will inherit custom properties \texttt{fnd:OA_TRANSLATABLE} from the TL EO that should have been set previously.

2. On the new TL VO, define the custom property at the VO level to \texttt{fnd:OA_TRANSLATION_TABLE = true}. The translated attributes in this new VO will inherit custom properties \texttt{fnd:OA_TRANSLATABLE} from the TL EO that should have been set previously.

3. Add \texttt{LanguagePEO} to the VO for Language and \texttt{SourceLang}. The \texttt{LanguagePEO} must be added to the model for the new TL VO for Language and \texttt{SourceLang}, using the 2 entity associations created above.

   a. Add \texttt{LanguagePEO} to the TL VO as an Entity Object, using the Entity Association created for the Language attribute.
b. Add LanguagePEO to the TL VO as an Entity Object again, using the Entity Association created for the SourceLang attribute.

4. Add Language and SourceLang display name attributes. Create 2 attributes in your VO for the Language and SourceLang display name, using the Description attribute from the LanguagePEO entity objects just added.

In the translation popup window, show the language description instead of the language code. For example, American instead of US.
Important:
When adding these attributes to the EO you must give them the alias `LanguageDisplayName` and `SourceLangDisplayName`, as shown below.

**Figure 16-32  Creating LanguageDisplayName Alias for Description Attribute**
The `LanguageDisplayName` and `SourceLangDisplayName` attributes should now be included in the TL VO.

Figure 16-34  Example Showing LanguageDisplayName and SourceLangDisplayName Included in TL VO
Important:

Set the Query Column Alias of the `LanguageDisplayName` attribute to "LANGUAGE_DISPLAY_NAME". The Translation Editor uses the Language Display Name to do the "order by" when displaying the translation data in the popup. You do not need to set the Query Column alias for `SourceLangDisplayName`, only `LanguageDisplayName`, as shown here.

**Figure 16-35  LanguageDisplayName Alias**

---

Important:

Set the Control hint label text of your `LanguageDisplayName` to "Language".
Set the Control hint label text of your `SourceLangDisplayName` to "Source Language."

---

5. Add a View Link for the VO. This new TL VO must be a child of the base VL VO. A View Link must be defined between the VL VO and the new TL VO.

### Add Translation VL to AM

1. Add Translation VO to the AM. Use this View Link to add the Translation VO to the AM, as a child of the VL VO.

2. Exclude this TLVO from loader VO list. If you are using the same model for both your UI and loader purpose, you will need to exclude this TLVO so that loader will skip this TLVO. For example, assume `StandardLookupTypeUIVO` is the loader VO and it now has a TLVO child. If you don’t exclude this child, the TLVO will appear in the loader file. What you do is to add `SD_EXCLUSION_LINKLIST` into your AM.xml file as shown here:

```xml
<CustomProperties>
  <Property
    Name= "SD_EXCLUSION_LINKLIST_StandardLookupTypeUI1"
    Value = "StandardLookupTypeUIVOToLookupTypeTL1"/>
</CustomProperties>
```
Important:

Note that in the above example, `StandardLookupTypeUI1` is the View Instance Name. And in the Value portion of `StandardLookupTypeUIVOToLookupTypeTL` is the View Link Instance Name. You can get the exact name in your AM > DataModel, as shown here.

Figure 16-36   StandardLookupType UI1 in Data Model

Figure 16-37   LookupTypeTL1 in Data Model

Your TL model should now be complete. Continue to the Design Time section to add the Translation Editor to your UI.

16.8.4.1.1 Design Time for Existing Application Tables

This section describes how to add a Translation Editor to an Application Table on an existing JSFF page. If you are creating a new Application Table, skip to the “Design Time for New Application Tables” section.

A design time wizard is used to add the Translation Editor to a setup UI.

For example, if you drag a TLVO Instance from the dataControl to your existing ApplicationTable > additionalToolbarButton > af:toolbar or ApplicationTable > f:facet-popup > panelGroupLayout then under Applications you can pick Translation Editor. You can drop the Translation Editor on either facet; the result will be exactly the same.

Note that you can not drag and drop directly onto ApplicationTable because it is not an ADF component that can be dropped.

This is an example drag and drop of your TLVO onto ApplicationTable > additionToolbarButton > af:toolbar.
You also can drag and drop your TLVO onto ApplicationTable > f:facet-popup > panelGroupLayout.

After the drag and drop, the wizard will show. The wizard will include input for a list of all the columns in the TL VO and you can remove the ones you do not want. The list will show all the attributes in the TL VO, except for WHO columns, LANGUAGE, SOURCE_LANG and the Primary Key. You can remove the translation attributes you do not want to display. You also can pick input text field or text region as the type. All other display properties (such as prompt) will be defaulted from control hints on the TL VO.
Figure 16-40  Create Translation Popup

When you click Continue, designer will add the popup code into your JSFF file. You should not edit anything in the code added to your page.

Turn On Auto Submit

Turn on the autoSubmit property for all your translation attributes so that any changes you make in your base table are immediately seen inside your Translation Editor popup table.

All updates you make inside the Translation Editor popup table will automatically be brought back to your base table.

Adjust your code accordingly if it is sensitive with autoSubmit turned on.

16.8.4.1.2 Design Time for Creating a New Application Table

The Application Table wizard lets the user choose whether or not to have the Translation Editor ability. The View Instances that will be available to enable for the Translation Editor will be those VOs marked with the fnd:OA_TRANSLATION_TABLE custom property. If there are no such VOs, the Translation Editor option will be disabled.
LookupTypeTL1 is shown automatically because it has defined the custom property \texttt{OA\_TRANSLATION\_TABLE = yes.}

Further steps are identical to the procedures in Design Time for Existing Application Tables.

**Information Written to the JSFF File**

Whether you add the Translation Editor feature for an existing application table or enable it when creating a new application table, Translation Editor writes two pieces of information into the JSFF page whether you create it within the Application Table wizard or you create it by dragging and dropping the TLVO into an existing application table.

This example shows information that will be written inside the JSFF file.

```xml
<fnd:applicationsTable tableId="ATt1" id="AT1"
..................................>
<af:table ............
<!-- Design time add the TL dialog id into your af:table partialTriggers !-->
  partialTriggers="............:::TlAfd1"

  <!-- TL button is placed inside ApplicationTable additionToolbarButtons -->
  <f:facet name="additionalToolbarButtons">
    <af:toolbar id="t1">
      <af:commandToolbarButton shortDesc="#{applcoreBundle.TRANSLATION_EDITOR}"
        icon="/images/applcore/patterns/language_qualifier_ena.png"
        hoverIcon="/images/applcore/patterns/language_qualifier_ovr.png"
        depressedIcon="/images/applcore/patterns/language_qualifier_dwn.png"
        disabledIcon="/images/applcore/patterns/language_qualifier_dis.png"
        disabled="#{backingBeanScope.TranslationEditorBean.checkNewRow.tlt2}"
        id="ctb1">
        <af:showPopupBehavior popupId="::TlAfp1"  <<---- Translation Editor popup id
          triggerType="action"/>
    </af:toolbar>
  </f:facet>
</fnd:applicationsTable>
```
Implementing the Translation Editor

</af:commandToolBarButton>
</af:toolbar>
</f:facet>

........................................
</f:facet>
</f:facet>
</af:table>

<!-- Design Time place the the TL popup dialog code at the end of your table -->
<f:facet name="popup">
<af:panelGroupLayout id="pgl2">
<af:popup contentDelivery="lazyUncached"
popupFetchListener="#{backingBeanScope.TranslationEditorBean.handlePopupFetch}"
popupCanceledListener="#{backingBeanScope.TranslationEditorBean.handlePopupCanceled}"
id="TlAfp1">
<af:dialog title="#{applcoreBundle.EDIT_TRANSLATIONS}"
cancelTextAndAccessKey="#{applcoreBundle.CANCEL}"
type="okCancel"
affirmativeTextAndAccessKey="#{applcoreBundle.OK}"
imodal="true"
dialogListener="#{backingBeanScope.TranslationEditorBean.updateBaseParent}"
id="TlAfd1">
<af:panelStretchLayout id="psl1" dimensionsFrom="children">
<f:facet name="center">
<af:table value="#{bindings.LookupTypeTL1.collectionModel}" var="row"
rows="#{bindings.LookupTypeTL1.rangeSize}"
emptyText="#{bindings.LookupTypeTL1.viewable ? applcoreBundle.TABLE_EMPTY_TEXT_NO_ROWS_YET : applcoreBundle.TABLE_EMPTY_TEXT_ACCESS_DENIED}" fetchSize="#{bindings.LookupTypeTL1.rangeSize}" rowBandingInterval="0" inlineStyle="width:840px; height:200px;"
selectedRowKeys="#{bindings.LookupTypeTL1.collectionModel.selectedRow}"
selectionListener="#{bindings.LookupTypeTL1.collectionModel.makeCurrent}" sortProperty="#{bindings.LookupTypeTL1.hints.LanguageDisplayName.name}" headerText="#{bindings.LookupTypeTL1.hints.LanguageDisplayName.label}" sortable="false" id="c19">
<af:column sortProperty="#{bindings.LookupTypeTL1.hints.Meaning.name}" sortable="false" headerText="#{bindings.LookupTypeTL1.hints.Meaning.label}" id="c21">
</af:column>
</af:table>
</f:facet>
</af:dialog>
</af:panelStretchLayout>
</af:popup>
</af:panelGroupLayout>
</af:dialog>
<af:column> and <af:inputText> Properties Copied from Base Attribute

If you have any properties set on the translatable attribute in the base table, design time will copy the same properties into the attribute inside the translation popup table.

These are the properties Designer will copy:

- `align`, `headerNoWrap`, `headerText`, `helpTopicId`, `minimumWidth`, `noWrap`, `rendered`, `shortDesc`, `showRequired`, `sortProperty`, `sortTable`, `visible`,...
16.8.4.1.3 Design Time for Adding Translation Editor into ADF Form

This is the start of your topic.

The Translation Editor feature is designed outside of the native ADF. Therefore, you must create your ADF form first and then drag and drop the Translation Editor view object onto the desire location in the ADF form.

If you have a simple form, you should drag and drop onto the applicationsPanel > actionButtonBar facet.

• Locate the Data Control created for the new Translation VO.
• Open the JSFF page. Drag-drop this data control onto the fnd:applicationsPanel > f:facet - actionButtonBar in the Structure window.
• In the Create menu, choose Applications > Translation Editor.

After the drag and drop, the Translation Editor wizard already described for the application table will open.

You also can choose to drag and drop the Translation Editor button inside your ADF form. For example:

Figure 16-43 Drag and Drop into ADF Form
**Note:**

Turn On Auto Submit.

Turn on the `autoSubmit` property for all of your translation attributes so that any changes you make in your base form will be seen immediately inside your Translation Editor popup table.

All updates you make inside the Translation Editor popup table will automatically be brought back to your base form.

Adjust your code accordingly if it is sensitive to having `autoSubmit` turned on.

This is the code written into your JSFF file:

```xml
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
    xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
    xmlns:f="http://java.sun.com/jsf/core"
    xmlns:c="http://java.sun.com/jsp/jstl/core"
    xmlns:fnd="http://xmlns.oracle.com/apps/fnd/applcore"
    xmlns:h="http://java.sun.com/jsf/html">
    <c:set var="profileuiBundle" value="#{adfBundle['oracle.apps.fnd.applicore.profiles.ui.CreateEditCategoryBundle']}" />
    <c:set var="profileuiMainBundle" value="#{adfBundle['oracle.apps.fnd.applicore.profiles.ui.ProfilesMainBundle']}"/>
    <af:pageTemplate viewId="/oracle/apps/fnd/applicore/templates/UIShellMainArea.jspx" id="pt1">
        <f:facet name="localArea">
            <fnd:applicationsPanel id="AP1" title="#{bindings.Name.inputValue == 'a' ? profileuiBundle.CREATE_PROFILE_CATEGORY : af:formatNamed(profileuiBundle.EDIT_PROFILE_CATEGORY, 'PROFILE_CATEGORY_NAME', bindings.Name.inputValue)}" scrollable="true" contentsFacet="scroll"
                navigationType="none" cancelVisible="true"
                cancelRendered="true" saveVisible="true"
                saveRendered="true" saveAndCloseVisible="true"
                saveAndCloseRendered="true"
                saveDisabled="#{bindings.CreatedBy.inputValue == 'SEED_DATA_FROM_APPLICATION' and securityContext.userName != 'SEED_DATA_FROM_APPLICATION'}"
                saveAndCloseDisabled="#{bindings.CreatedBy.inputValue == 'SEED_DATA_FROM_APPLICATION' and securityContext.userName != 'SEED_DATA_FROM_APPLICATION'}">
                <f:facet name="contents">
                    <h:panelGroup id="pg1">
                        <af:panelGroupLayout id="pg1">
                            <af:panelHeader text="#{profileuiBundle.PROFILE_CATEGORY}" id="ph1">
                                <!-- Design time will write a partial trigger inside your form to point to the TL dialog !-->
                                <af:panelFormLayout maxColumns="1" rows="1" id="pf12" partialTriggers="TlAfd1"/>
                            </af:panelHeader>
                            <!-- Your existing af:form content -->
                        </af:panelGroupLayout>
                        <h:panelGroup id="pg1">
                            <af:panelGroupLayout id="pg1">
                                <af:panelHeader text="#{profileuiBundle.PROFILE_CATEGORY}" id="ph1">
                                    <!-- Design time will write a partial trigger inside your form to point to the TL dialog !-->
                                    <af:panelFormLayout maxColumns="1" rows="1" id="pf12" partialTriggers="TlAfd1"/>
                                </af:panelHeader>
                            </af:panelGroupLayout>
                        </h:panelGroup>
                    </h:panelGroup>
                </f:facet>
            </fnd:applicationsPanel></f:facet>
        </af:pageTemplate>
    </jsp:root>
```
Implementing the Translation Editor

<!-- could also put the TL button at the end of your form -->
<af:panelFormLayout>
<af:panelHeader>
<af:panelGroupLayout>
<!-- The TL popup dialog code starts here -->
<af:popup id="TlAfp3" contentDelivery="lazyUncached"
popupFetchListener="#{backingBeanScope.TranslationEditorBean.handlePopupFetch}"
popupCanceledListener="#{backingBeanScope.TranslationEditorBean.handlePopupCanceled}"
>
<af:dialog id="TlAfd1"
title="#{applcoreBundle.EDIT_TRANSLATIONS}"
cancelTextAndAccessKey="#{applcoreBundle.CANCEL}"
type="okCancel"
affirmativeTextAndAccessKey="#{applcoreBundle.OK}"
modal="true"
dialogListener="#{backingBeanScope.TranslationEditorBean.updateBaseParent}"
>
<af:panelStretchLayout dimensionsFrom="children"
id="psl3">
<f:facet name="center">
<af:table
value="#{bindings.ProfileCategoryTLUI1.collectionModel}"
<!-- The TL table content is the same as the above example -->
</f:facet>
</af:panelStretchLayout>
</af:dialog>
</af:popup>
</af:panelGroupLayout>
</af:panelFormLayout>

<!-- The TL button is placed inside Application Panel actionButtonBar -->

<h:panelGroup id="pg2">
<af:commandToolbarButton shortDesc="#{applcoreBundle.TRANSLATION_EDITOR}"
icon="/images/applcore/patterns/ipaddress_ena.png"
hoverIcon="/images/applcore/patterns/ipaddress_ovr.png"
depressedIcon="/images/applcore/patterns/ipaddress_dwn.png"
disabledIcon="/images/applcore/patterns/ipaddress_dis.png"
id="ctb1"
disabled="#{backingBeanScope.TranslationEditorBean.checkNewRow.tlt29}"
>
<af:showPopupBehavior popupId="TlAfp1"
triggerType="action"/>
</af:commandToolbarButton>
</h:panelGroup>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</f:facet>
</af:message id="msg1"
If you have a more complicated form that involves multiple regions (multiple TL ViewObject), you should drag and drop to the location that makes the most sense to you.

For example, if you have five fields on your form, A, B, C, D, E and if you drag and drop the TL ViewObject onto attribute C, the Translation Editor button will appear after attribute C.

16.8.4.1.4 Removing the Translation Editor

The two pieces of information written to your page file are the Translation Editor button and the dialog. You can search for “TlAfp” to find the button and dialog locations in your page so that you can remove them manually.

If you add a new translation attribute into your TLVO and you would like to include this new attribute, you can remove them. Then you can drag and drop the TLVO again so that the new attribute will be included.

16.9 About the Create Page Wizard

If the implementing code is in place, Create Page lets a user create, text, and export a new application page without having to use JDeveloper. This is useful, for instance, in creating a new homepage, testing it, and finally moving it from one system to another, such as from a test environment to a production environment.

The basic steps to create a page are:

1. Create and then enter a sandbox. Using a sandbox is described in the Oracle Sales Cloud Customizing Sales guide.
2. In the Navigator menu, select New Page.
3. Complete the Create Page wizard.
4. Edit and test the page.
5. Publish the page.

16.9.1 How to Complete the Create Page Wizard

When you click New Page on the Navigator menu, the Create Page wizard displays.
Name: Enter a name for the page.

Menu Category Title: Enter a menu category title. The page name will be displayed under this title.

Icon: Click the Search icon to select an icon to represent this page. This step is mandatory and the icon must be selected from the choices presented; a user-supplied icon cannot be used.

Web Page: [Optional] Enter the URL of the application that you wish to host in this page in an iFrame. For example, if you want to run a calculator in your page, you can enter this URL:

http://www.embedcalculator.com/flash/StandardCalculator.swf

This usually is used to create a page where content is coming from another application. This can be a link to a related application, or to a public web site, as shown in the calculator example.

Secure Token: If you need to add a secure token to the URL, click the key icon.

This option is used when you link a menu item to a secure token URL of an application outside Fusion Applications.

An HTTPS protocol is required to access the application.

The secure token expires if the user session is inactive. Refresh the page to regenerate the tokens.

This additional dialog displays:

Web Application: This is a dropdown menu that shows third-party applications that have been registered in the topology tables. To register an application, use the Register Enterprise Applications task in Setup and Maintenance. Registered content usually includes the protocol (such as http or https), the host, port and context root. This means that the beginning section of the URL can be built from this information.

Destination for Web Application: The destination is anything else to be appended on the URL.

Secure Token Name: [Optional] This can be any name you choose.
For example:

Web Application = "BlueApp"
Destination for Web Application = "/faces/showItem.asp" - The beginning "/" is optional, the design time will insert it if it is not included.
Secure Token Name = "userToken"

When saved, the design time will create a menu node with just a destination field with the following:

destination="#{EndPointProvider.externalEndpointByModuleShortName['BlueApp']}/faces/showItem.asp?userToken=#{applCoreSecuredToken.trustToken}"

At runtime, this could evaluate to:

https://BlueAppsHost:9001/BlueApp/faces/showItem.asp?
userToken=9439439432JHU8I53KL88E3

Click Save and Close.

Click Save and Continue.

16.10 Troubleshooting UI Shell Issues

This section presents the more-common problems, with their solutions, that may be experienced by Oracle Fusion applications developers. Also see "Troubleshooting Customizations" in the Extensibility Guide for Developers.

16.10.1 ApplSession Is Not Created

Problem

When ApplSession is not created, these problems occur:

• User Language settings are not set.
• Data security policies are not applied.
• Webservice requests are run as anonymous users instead of named users.

Solution

To resolve this problem:

• Check that the JpsFilter section is correct in web.xml: Applcore Session code depends on the JPS subject. The entry must appear exactly as shown in Example 16-16.

• Check that the ApplSessionFilter section is correct in web.xml: ApplSession must be configured immediately after JpsFilter. The entry must appear exactly as shown in Example 16-16.

• If the issue is with the ApplSession not being created over a web service request using SOA, check how the ApplSession context interceptor is configured.

ApplSession information is preserved over SOA/Web Service requests with context interceptors that fire when requests are sent and received. The ApplSessionContext interceptor, in particular, will handle propagating all session attributes from the caller to the web service, and will even attempt to reuse the same session Id if running against the same database. To enable the context
interceptor, the `weblogic-application.xml` on both the caller and the called should include the `oracle.applcore.config` shared library:

```xml
<library-ref>
    <library-name>
        oracle.applcore.config
    </library-name>
</library-ref>
```

If the context interceptor has not been enabled, the `prepareSession()` of the root application module call should still be able to create the session as the right user, so long as the Subject has been established correctly in the Oracle Web Services Manager layer. But in that case, no other session attributes will be propagated, and the session cannot be reused.

- Obtain the session Id from the cookie. The cookie is located in the browser's cookies folder.
  
  The session cookie is obtained so that subsequent queries can be run to verify the session's existence and obtain additional info about the session. Note that it is the session cookie that is stored in the browser cookie location, not the session Id. They represent the same session, but the actual session Id is internal.
  
  Cookie name: `ORA_FND_SESSION_<db_instance_name>`

  The session cookie value will appear similar to: `DEFAULT_PILLAR:wvlul8CU14FJyjySCPQtwaePhBOyiUT2wexYdIYXh5VwV9i9koUL8L3Qhm0bNrZ8:1282160020177`

  To retrieve the actual session id, run this SQL statement: `select session_id from fnd_sessions where session_cookie=value_obtained_from_cookie_value`.

  **Note:** The `value_obtained_from_cookie_value` is the alpha-numeric string between the two colons. In the example, it is `wvlul8CU14FJyjySCPQtwaePhBOyiUT2wexYdIYXh5VwV9i9koUL8L3Qhm0bNrZ8`. This value will always be different.

**Example 16-16  ApplSessionFilter and JpsFilter Configuration in web.xml**

```xml
<filter-mapping>
    <filter-name>JpsFilter</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
<filter-mapping>
    <filter-name>ApplSessionFilter</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

### 16.10.2 Navigator Shows a Little White Box

**Problem**

Clicking the Navigator in the global area shows a little white box.

This is an issue that developers would see.
Solution when in a production environment

- Make sure that you are signed in as a user that has access to some pages in the Navigator menu. You can type the full URL of the page directly into the browser and see if the user can access it.

- Make sure there is a valid connection to the LDAP server. This is needed to check the policies of each entry in the Navigator menu.

- Use Enterprise Manager to make sure the Applications are registered in the topology tables.

- Make sure the Applications Core Setup application is deployed. (From JDeveloper, run .../atgpf/applcore/applications/FndSetup/FndSetup.jws.) This contains the Navigator menu file and usually is deployed to the Oracle Fusion Applications Global domain.

- Check that the menus are in MDS. Check for entries with the path oracle/apps/menu.

Solution when in a development environment

- You usually should run without menu security. The first thing to check is if you have started your web server with this parameter to turn off security checking:
  
  -DAPPLCORE_TEST_SECURED_MENU=N

- Then check steps 3 through 5 in the solution for a production environment.

16.10.3 Navigator Shows Unfiltered Entries

Problem
The Navigator menu is showing all entries, rather than being filtered by what the user has access to, and the Welcome page has all tabs exposed.

Solution
This is caused by menu security being turned off. This should happen only in development environments.

Because menu security is on by default, check that the web server was not started with -DAPPLCORE_TEST_SECURED_MENU=N.
This chapter discusses the Applications Tables, Trees and Tree Tables components used to implement user interface features in JDeveloper.

This chapter includes the following sections:

- Implementing Applications Tables
- Implementing the Applications Tree
- Implementing Applications Tree Tables
- Using the Custom Wizard with Applications Popups

For basic information, see:

- Getting Started with Your Web Interface
- Implementing the UI Shell
- Implementing Applications Panels_Master-Detail_Hover_and Dialog Details

17.1 Implementing Applications Tables

Applications tables are UI components that already contain an Oracle ADF table, a menu bar, a toolbar, and related popups. Developers do not need to create and assemble all these components separately.

Tables include the following:

- Table toolbars with default buttons
- Elements, such as an ADF table and custom toolbar buttons
- Default table actions
- Create actions, such as Create Inline and Create a Duplicate.
- Edit actions, such as Table Row Edit (Dialog Window) and Table Row Edit (Page)

You must use Applications tables to standardize layout and appearance consistency for all your page tables, including read-only pages. When you create an Applications table, you can add table components that allow users to select the table’s contents.

When you add an Applications Table to your JSF page, components of the Applications Table, such as the ADF table, are bound to the model.
Before you begin:
Before you can use Applications tables, you must be familiar with JDeveloper and be able to create JSF pages.

17.1.1 Understanding Applications Tables Facets and Properties

Each table has properties and facets. Properties include table qualities, such as the unique identification number and the type of pattern exposed in the table. Facets are locations for table data, such as locations where you can add toolbars or menu bars. This section describes Applications table properties and facets.

Note:
Any buttons or menu items added in a facet render with a separator because adding more than one component to a facet requires having `<af:group>` around the element. By default, having a group component introduces a separator as an ADF rule.

Table 17-1 describes Applications table facets and facet contents.

<table>
<thead>
<tr>
<th>Facet Name</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>table</td>
<td>Holds ADF table.</td>
<td>ADF Table</td>
</tr>
<tr>
<td>additionalToolbarButtons</td>
<td>Holds toolbar buttons.</td>
<td>ADF command toolbar buttons under an ADF toolbar</td>
</tr>
<tr>
<td>additionalActionItems</td>
<td>Holds menu items to be added to default Action pulldown menu items.</td>
<td>ADF menu item component</td>
</tr>
<tr>
<td>appsTableSecondaryToolbar</td>
<td>Facet for adding more commandToolbar button components to secondary toolbar.</td>
<td>ADF Command Toolbar Buttons</td>
</tr>
<tr>
<td></td>
<td>(Adds more icons.)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Icons usually perform the same actions as menus, but you put the most common as icons so you do not need to pull down the menu.</td>
<td></td>
</tr>
<tr>
<td>appsTableStatusBar</td>
<td>Holds components that contain status bar items. These status bar items are merged with standard items provided by the panelCollection property.</td>
<td>ADF menu item component</td>
</tr>
<tr>
<td>appsTableViewMenu</td>
<td>Holds menu items to be added to default View pulldown menu items of the panelCollection. To add multiple menu items to the view menu, add <code>af:group</code> components containing <code>af:menuItems</code>.</td>
<td>ADF menu item component</td>
</tr>
</tbody>
</table>
Table 17-1  (Cont.) Applications Table Facets

<table>
<thead>
<tr>
<th>Facet Name</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>appsTableAfter</td>
<td>Facet for adding more commandToolbar button components after toolbar. In this facet, any toolbar buttons added appear in a separate row below the normal group of toolbars.</td>
<td>af:toolbar or af:groups of af:toolbars</td>
</tr>
<tr>
<td>Toolbar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>popup</td>
<td>Holds popups. See Using the Custom Wizard with Applications Popups.</td>
<td>Popups under a layout component.</td>
</tr>
<tr>
<td></td>
<td>Important: When a popup is used to create or duplicate a row in an Applications Table, you need to write your own logic behind the popup's Cancel button (Action/ ActionListener) to remove the newly-created row. This can be done by either:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• A managed bean method that removes the newly-created row.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Setting the Cancel button's Action property to the rollback method defined in the pageDef file. This method would be defined in the pageDef file if the &quot;rollback&quot; from the operations of the dataControl is dragged and dropped onto the page.</td>
<td></td>
</tr>
</tbody>
</table>

Table 17-2 describes Applications table properties (including properties that are part of the default managed bean), their allowable values, and default actions.

Table 17-2  Applications Table Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Unique identification number for this applications table.</td>
<td>string</td>
</tr>
<tr>
<td>rendered</td>
<td>Whether the applications table is rendered (that is, converted from an object-based description into a graphical image for display).</td>
<td>boolean expression</td>
</tr>
<tr>
<td>tableId</td>
<td>Unique identification number of the underlying ADF table corresponding to this applications table.</td>
<td>string</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Values</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>createPatternType</td>
<td>Whether a Create pattern is enabled and, if so, which pattern. User action: Click Create.</td>
<td><code>&lt;default&gt;</code>, <code>inline</code>, <code>secondaryWindow</code>, <code>page</code>&lt;br&gt;<strong>&lt;default&gt;:</strong> No rows are created. You might choose this value if your table is read-only.&lt;br&gt;<code>inline</code>: New row is created at the top of the current table. If you choose this value, the createpartialTriggers property on the ADF table is set automatically.&lt;br&gt;<code>secondaryWindow</code>: Popup is displayed, allowing users to enter values into a new table row. The new row is added to the top of the table. If you choose this value, you must also set the corresponding Create Popup Id. In addition, you must create the popup UI that shows input fields.&lt;br&gt;<code>page</code>: An ADFc Controller outcome is returned such that navigation to the next view activity occurs.</td>
</tr>
<tr>
<td>editPatternType</td>
<td>Whether an Edit pattern is enabled and, if so, which pattern. User action: Click Edit.</td>
<td><code>&lt;default&gt;</code>, <code>secondaryWindow</code>, <code>page</code>&lt;br&gt;<strong>&lt;default&gt;:</strong> No rows become editable. You might choose this value if your table is read-only.&lt;br&gt;<code>secondaryWindow</code>: Popup is displayed, allowing users to edit values in the currently selected table row. If you choose this value, you must also set the corresponding Edit Popup Id.&lt;br&gt;<code>page</code>: A standard outcome is returned. In this case, users can edit the values in the currently selected table row.</td>
</tr>
<tr>
<td>duplicatePatternType</td>
<td>Whether a Duplicate pattern is enabled and, if so, which pattern. This pattern lets you create an object by duplicating an existing object. The duplication helps you by pre-filling some values. You have full control and can change any of the values during the creation process. User action: Click Duplicate.</td>
<td><code>&lt;default&gt;</code>, <code>inline</code>, <code>secondaryWindow</code>, <code>page</code>&lt;br&gt;<strong>&lt;default&gt;:</strong> No rows are duplicated.&lt;br&gt;<code>inline</code>: Selected row is duplicated. If you choose this value, you must also set the partialTriggers property on the ADF table to:&lt;br&gt;<code>&lt;Applications_Table_Id&gt;:duplicate</code>&lt;br&gt;<code>secondaryWindow</code>: Popup is displayed, allowing users to duplicate the currently selected table row but not its contents. If you choose this value, you must also set the Duplicate Popup Id.&lt;br&gt;<code>page</code>: Currently selected row is duplicated, but not the values it contains. If the applications table is part of an applications panel, set the partialTriggers property on the ADF table to:&lt;br&gt;<code>&lt;Applications_Panel_Id&gt;:&lt;Applications_Table_Id&gt;:duplicate</code></td>
</tr>
<tr>
<td>deleteEnabled</td>
<td>Whether a Delete pattern is enabled. User action: Click Delete.</td>
<td><code>boolean</code>&lt;br&gt;Selected row is deleted.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Values</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>createActionListener</td>
<td>Action binding for the Create button.</td>
<td>method expression</td>
</tr>
<tr>
<td>editActionListener</td>
<td>Action binding for the Edit button.</td>
<td>method expression</td>
</tr>
<tr>
<td>duplicateActionListener</td>
<td>Action binding for the Duplicate button.</td>
<td>method expression</td>
</tr>
<tr>
<td>deleteActionListener</td>
<td>Action binding for the Delete button.</td>
<td>method expression</td>
</tr>
<tr>
<td>createPopupId</td>
<td>Id assigned to the popup to be invoked when users click the Create button.</td>
<td>string</td>
</tr>
<tr>
<td>editPopupId</td>
<td>Id assigned to the popup to be invoked when users click the Edit button.</td>
<td>string</td>
</tr>
<tr>
<td>duplicatePopupId</td>
<td>Id assigned to the popup to be invoked when users click the Duplicate button.</td>
<td>string</td>
</tr>
<tr>
<td>createText</td>
<td>Value that overrides the default label for Create menu item. It also will be shown as the short description for the Create button.</td>
<td>expression</td>
</tr>
<tr>
<td>editText</td>
<td>Value that overrides the default label for Edit menu item. It also will be shown as the short description for the Edit button.</td>
<td>expression</td>
</tr>
<tr>
<td>duplicateText</td>
<td>Value that overrides the default label for Duplicate menu item. It also will be shown as the short description for the Duplicate button.</td>
<td>expression</td>
</tr>
<tr>
<td>deleteText</td>
<td>Value that overrides the default label for Delete menu item. It also will be shown as the short description for the Delete button.</td>
<td>expression</td>
</tr>
<tr>
<td>attachText</td>
<td>Value that overrides the default label for Attach menu item. This attribute is deprecated.</td>
<td>expression</td>
</tr>
</tbody>
</table>
### Table 17-2 (Cont.) Applications Table Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>featuresOff</td>
<td>List of default features to turn off for the panelCollection</td>
<td>string</td>
</tr>
<tr>
<td>inlineStyle</td>
<td>The CSS styles to use for the panelCollection component inside Applications Table component. This is intended for basic style changes. *Note: Do not set the width using the inlineStyle attribute on either Applications Table or panelStretchLayout. Applications Table can be stretched by placing it in the center facet of an ADF panelStretchLayout component.</td>
<td>string</td>
</tr>
<tr>
<td>styleClass</td>
<td>styleClass to use for the panelCollection component inside Applications Table component.</td>
<td>string</td>
</tr>
<tr>
<td>exportEnabled</td>
<td>Rendered attribute for Export button and menu item.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>createImmediate</td>
<td>Sets immediate attribute value of &quot;Create&quot; toolbar button and &quot;Create&quot; menu item.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>deleteImmediate</td>
<td>Sets immediate attribute value of &quot;Delete&quot; toolbar button and &quot;Delete&quot; menu item.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>duplicateImmediate</td>
<td>Sets immediate attribute value of &quot;Duplicate&quot; toolbar button and &quot;Duplicate&quot; menu item.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>editImmediate</td>
<td>Sets immediate attribute value of &quot;Delete&quot; toolbar button and &quot;Delete&quot; menu item.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>attachImmediate</td>
<td>Sets immediate attribute value of &quot;Attach&quot; toolbar button and &quot;Attach&quot; menu item.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>primaryToolbarRendered</td>
<td>Sets the rendered attribute value of the primary toolbar. When Create, Duplicate, Update, Delete actions, attach, export are not turned on, the primaryToolbarRendered should be set to false so that an empty toolbar will not be displayed.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Values</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>secondaryToolBarRendered</td>
<td>Sets rendered attribute value of the secondary toolbar. When no af:commandToolbarButton is added to appsTableSecondaryToolbar facet, secondaryToolBarRendered should be set to false so that an empty toolbar will not be displayed.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>&lt;button_name&gt;PartialTriggers</td>
<td>Partial triggers attribute for the &lt;button_name&gt; toolbar button. For example: deletePartialTriggers</td>
<td>String of IDs. Important: The PartialTriggers attribute must be entered manually by the developer. This is because, at design time, the JDeveloper Property Inspector can: • Select the incorrect ID. • Append square brackets around the selected id, such as [id1 id2]. Example 1: To disable the Edit, Delete and Duplicate buttons when the table is empty, set this property on the editDisabled, deleteDisabled or duplicateDisabled property of the Applications Table. #{bindings.VOiterator.estimatedRowCount == 0 ? true: false} where VOiterator is your iterator name Example 2: Disable any of the buttons in the Applications Table according to the functional rules or by setting disable=false when create is selected on an empty table (considering these buttons were disabled following Example 1). To do this, create an attribute binding on the view object attribute that will decide whether the row can be deleted/edited/duplicated. For example, you can use a binding similar to this example on the disable property of a button: #{bindings.MyAtttrBinding.inputValue == 'compare value' ? true : false} Add Partial Page Refresh (PPR) on the button to the table ID of the af:table. This does not require any change in the selectionListener of the table. Keep the default one.</td>
</tr>
<tr>
<td>createAction</td>
<td>Action binding for the Create icon.</td>
<td>method expression</td>
</tr>
<tr>
<td>editAction</td>
<td>Action binding for the Edit icon.</td>
<td>method expression</td>
</tr>
<tr>
<td>duplicateAction</td>
<td>Action binding for the Duplicate icon.</td>
<td>method expression</td>
</tr>
</tbody>
</table>
Table 17-2  (Cont.) Applications Table Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteAction</td>
<td>Action binding for the Delete icon.</td>
<td>method expression</td>
</tr>
<tr>
<td>createEnabled</td>
<td>Rendered attribute for create</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>duplicateEnabled</td>
<td>Rendered attribute for duplicate</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>editEnabled</td>
<td>Rendered attribute for edit</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>createPartialSubmit</td>
<td>PartialSubmit attribute for create</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>createDisabled</td>
<td>Disabled attribute for create</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>editDisabled</td>
<td>Disabled attribute for edit</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>duplicateDisabled</td>
<td>Disabled attribute for duplicate</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>deleteDisabled</td>
<td>Disabled attribute for delete</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>confirmDelete</td>
<td>Set this value if you want delete confirmation to be displayed.</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td></td>
<td>The default message is <strong>The selected record(s) will be deleted. Do you want to continue?</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>To change this, use the <code>deleteMsg</code> attribute.</td>
<td></td>
</tr>
<tr>
<td>deleteMsg</td>
<td>Provide a customized delete confirmation message that can be shown in the popup.</td>
<td>String value or Expression Language expression</td>
</tr>
<tr>
<td>actionsMenuRendered</td>
<td>Rendered attribute for Actions menu</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>actionsContentDelivery</td>
<td>ContentDelivery attribute for Actions menu. This attribute can take two values.</td>
<td>String value</td>
</tr>
<tr>
<td></td>
<td>• lazy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• immediate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The default value is <strong>immediate</strong>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Setting the attribute value to <strong>lazy</strong>:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Provides better performance.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Does not allow for enabling/disabling of the menu items based on client actions, such as current row selected.</td>
<td></td>
</tr>
<tr>
<td>toggleEditRendered</td>
<td>The toggleEditRendered feature is used to render the editAll or clickToEdit choices for Applications Table. See <strong>Toggle Click to Edit / Edit All in Applications Table.</strong></td>
<td>Boolean. Default value is <strong>False</strong>.</td>
</tr>
</tbody>
</table>
**Table 17-2 (Cont.) Applications Table Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>exportRowLimit</td>
<td>This property sets the row limit for exported rows when &quot;Export to Excel&quot; is used to export records to Excel. If this is not set, then the row limit is picked from the value that is set for the profile option FND_EXPORT_ROWLIMIT.</td>
<td>Positive integer.</td>
</tr>
<tr>
<td>defaultToolbar</td>
<td>Use to hide the DETACH/FREEZE/WRAP labels and just display the related icons in the toolbar to reclaim space used by the display of the labels on the table's panel collection.</td>
<td>iconAndText will display the labels. icon will hide the labels.</td>
</tr>
</tbody>
</table>

**Note:**
If you choose secondaryWindow as the pattern type for any property, and you have set the popup Id for that button, selecting the button invokes the popup.

**Model**
The Applications Table does not expose any bindings to the model. However, components within the Applications Table, like the ADF table, will be bound to the model.

**Controller**
The Applications Table component ships a default managed bean that performs the following functions that only work with rowSelection="single" on the ADF table:

- Default event handlers for all toolbar button action events. Event handler delegates to custom action method if set on the button action property.
  - A new row is added into the table when the Create icon is clicked, and the Create Pattern Type is inline.
  - A new row is added into the table and a popup is invoked with the newly-created row available for inserting values (the UI for the popup to show input fields for the new row has to be created separately by the developer), when the Create Pattern Type is secondaryWindow.
  - A new row is not added when the Create Pattern Type is Page. The developer is responsible for wiring the navigation to the page when the icon or menu item is clicked. Only a standard outcome is returned from the default handler. The developer could use this default outcome to define a navigation rule.
  - The selected row is made available for editing in a popup when the Edit icon is clicked, and Edit Pattern Type is secondaryWindow.
– When the Edit icon is clicked, and Edit Pattern Type is Page, only a standard outcome is returned.

– The Duplicate icon is handled the same way as Create. All attribute values except the primary key values are duplicated.

– Clicking the Delete icon deletes the selected row.

• If the secondaryWindow option is chosen for any pattern type, and the corresponding popup Id is set for that button (mandatory), selecting the button invokes the popup.

• If Page is chosen for any pattern type, a standard outcome is returned on clicking the button. Standard outcomes are create, edit, duplicate and delete for the four respective toolbar buttons.

To allow Applications developers access to some of the implementation, the Applications Table exposes a public class oracle.apps.fnd.appcorepatterns.ApplicationsTableEventHandler that contains default event handlers for all the buttons. The button methods are named as process<buttonName>, such as processCreate and processEdit. Application developers writing custom action handlers can also use the default implementation by calling these methods.

Use

For example, to attach a custom button handler to the Create button, follow these steps:

1. Define a managed bean class, as shown in Example 17-1.

2. Register the managed bean in the faces-config of the project.

3. Bring up the Property Inspector for the Applications Table, and choose the Create Action property. Set #{CustomEventHandler.processCreate} as the expression for the property.

Example 17-1  Defining a Managed Bean Class to Attach a Custom Handler to a Button

```java
import oracle.apps.fnd.appcorepatterns.ui.ApplicationsTableEventHandler;
import oracle.apps.fnd.appcorepatterns.ui.util.PatternUtils;

public class CustomEventHandler
{
    public String processCreate()
    {
        // Custom code
        ...

        // Call default event handler if required. It will return a standard outcome for this button click.
        ApplicationsTableEventHandler appTableHandler = ApplicationsTableEventHandler.getInstance();
        String outcome = appTableHandler.processCreate();

        // If popup is required to be invoked after event handling
        PatternUtils.invokePopup(popupId);

        return outcome;
    }
}
```
17.1.2 How to Create an Applications Table

You can create and add Applications tables to pages or page fragments. Using the wizard will create a working table without you having to hand-code every step. When the table is created, you can change any parameters from the Property Inspector.

17.1.2.1 Add Applications Tables to JSF Pages or Page Fragments

You create Applications tables in the Applications Tables wizard, which is displayed when you add the tables to your JSF pages (or page fragments) from the Component Palette or the Data Controls panel.

To start the Applications Table wizard from the Component Palette:

1. Open the Component Palette.
2. In the list, choose Applications.
3. In the list, click Table. JDeveloper will attempt to place the table at the current cursor location. If the current location is not appropriate, an error message displays. You also can drag the Table icon to the page in either the Design or the Source view. A plus + sign will be added to the arrow when it is over a location in which a table can be inserted.

The Applications Table wizard is displayed.

To start the Applications Table wizard using the Data First method:

1. In the Application Navigator, open the Data Control panel.
2. Navigate to the data source that you want to bind to the Applications table. The data source must represent a rowset; that is, it must be a view object.
3. Drag and drop the data control to the JSF page.
4. In the Create context menu that is displayed, choose Applications > Table.

The Applications Table wizard is displayed.

17.1.2.2 Add Applications Table Components Using the Applications Table Wizard

This section explains how to use the Applications Table wizard to add components to your table.

In the Applications Table wizard you can:

- Bind your table to a data source
- Create placeholder columns and define their attributes
- Enable table ADF behaviors
- Select table default actions

The Applications Table wizard has five main dialogs. Click Cancel in any dialog to cancel your actions and exit the wizard. Click Next to accept your settings and continue.
To add an Applications Table using the Applications Table wizard:
When the Applications Table wizard is launched, the Specify the data source ... dialog is displayed.

Figure 17-1  Create Applications Table Dialog (Step 1 of 5)

In this dialog:

- Bind data to the table (optional):
  - Select the Bind Data Now box to bind a data control to the table.
  - In the Data Source field, click Browse to choose from a list of data sources available for binding.
    This step might take a few minutes.
    The Select Table Data Collection dialog is displayed.
Select the data source to bind to your table and click **OK**.

If you do not bind the data, the table creates placeholder columns that can be used for layout purposes.

Click **Next** to display the Set Row Selection and Layout Columns dialog.
Select Read-only Table to prevent users from modifying the data. If you select Read Only, the options in the Component To Use column will change from Input to Output components.

When you choose a data source to bind to your table, these options become available.

**Row Selection**

Select None to disable row selection by users.

Select Single to allow users to be able to select individual rows in the table. This will set the rowSelection attribute to single. Selecting this option means that instead of the UI component determining the selected row, the iterator binding will access the iterator to determine the selected row. This is recommended when using ADF Model data binding.
Select **Multiple** to allow users to be able to select multiple table rows.

**Sorting**

Select to allow users to be able to sort columns. Selecting this option means that the iterator binding will access the iterator which will perform an order-by query to determine the order. This is recommended when using ADF Model data binding. Only keep this checkbox unselected if you do not want to allow column sorting.

**Filtering**

Select to allow users to be able to filter the table based on given criteria. Selecting this option allows the user to enter criteria in text fields above each column. That criteria is then used to build a query-by-example search on the collection, so that the table will display only the results returned by the query.

- The **Columns** section is used to set the behavior of the table's columns.

  **Group**

  Select two or more columns then click this link to group the columns together in the table. The selected columns will be grouped together under a parent column.

  **Ungroup**

  Select columns that are grouped then click this link to ungroup the columns.

**Display Label**

By default, the label is bound to the labels property for the attribute on the table binding. You can instead enter text or an Expression Language expression to bind the label value to something else, for example, a key in a resource file.

**Value Binding**

Shows the attribute to which the value is bound. Use the drop-down list to choose a different attribute. If you simply want to rearrange the columns, you should use the order buttons. If you do change the attribute binding for a column, the label for the column also changes.

**Component to Use**

Shows the component used to display the value. Use the drop-down list to choose a different component. By default, output text components are selected for read only tables. Input text components are selected for all other tables. Input date components are used for attributes that are dates. If you want to use a different component, such as a command link or button, you need to use this dialog to select the outputText component, and then in the Structure window, replace the component with the desired UI component (such as a command link). By default, only ADF Faces components are shown in the menu. You can allow JSF Implementation components to also be chosen.

**Add Column**

Select a column name from the attributes list and click + to add the column name to your table. Repeat this step for all your table column names.

**Delete Column**

Click X to delete a column name.

**Sort Column Order**

Click the up or down arrows to sort the order of the columns.
Click **Next** to display the Select Table CRUD Patterns (Create, Read, Update and Delete) dialog.

**Figure 17-4   Select table CRUD Patterns Dialog (Step 3 of 5)**

Select default actions for your Applications table (optional):

- **Create / Pattern**: Creates a table row.
  
  When you have chosen to enable row creation, choose a pattern from the list to invoke an action.
  
  - **Inline**: Perform the action on the current table row (only available for the Duplicate action)
  
  - **Popup**: Bring up a div modal window on top of the current page for the requested action
If you choose the Popup pattern, click **Configure Popup** to display the Applications Popup wizard (see Using the Custom Wizard with Applications Popups) and follow the instructions to configure the popup associated with this pattern. See Understanding Applications Tables Facets and Properties for important information about the popup’s **Cancel** button.

- **Page** - Replace the current page or page fragment with a completely separate page or page fragment to perform the action. Page fragments are used when using bounded task flows.

- **Duplicate / Pattern**: Duplicates the row.
  
  When you have chosen to enable row duplication, choose a pattern from the list to invoke an action.

- **Edit / Pattern**: Enables row modification.
  
  When you have chosen to enable row editing, choose a pattern from the list to invoke an action.

- **Delete**: Allows users to delete the row.
  
  **Confirm Delete**: Select this option so that the default **The selected record(s) will be deleted. Do you want to continue?** prompt displays in a popup when the delete row function is used.
  
  When you set the `confirmDelete` attribute to true, the confirmation popup displays and the row is deleted when you click **Yes**. For this to work correctly, the `partialTriggers` on the `af:table` inside the `fnd:applicationsTable` should include `::confirm`, and the `::delete` and `::deleteConfirm` ids must be removed so the `partialRefresh` happens only when you click **Yes** in the popup. Developers can choose to set the immediate property on the **Yes** button by using the `deleteImmediate` attribute. The **No** button has immediate set to true by default. See Manually Enabling Delete Confirmation.

  **Confirmation Message**: If you want to replace the default confirmation message with a custom one, enter the string here. The string will be converted to a text resource and added to the default resource bundle.
  
  If you already have a confirmation message defined in a resource bundle, click the ellipsis and choose from the list.
  
  If you need to manually create or edit the Delete Confirmation parameter, see "Manually Enabling Delete Confirmation."

- **Export**: Export the data to a Microsoft Excel-compatible file.

For more information on pattern types, see Table 19-2, "Applications Table Properties".

Click **Next** to display the Set Table Features dialog.
This dialog allows selection of options to simplify the output.

**Table Summary:** Enter a summary description of the table. The attribute is written to the `af:table` as:

```javascript
summary='#{viewControllerBundle.SUMMARY_TEXT_RESOURCE}'
```

The default selection, **Use Default Configuration**, selects the most simplified UI output.

The business rules on what each of these options render to the page are explained below. Refer to the callout numbers (1–9) in the image above.

1. **Show Actions Menu:** Sets `fnd:applicationsTable actionsMenuRendered = "false"` when it is not selected. Nothing is written when it is selected.

2. **Show View Menu:** Adds `fnd:applicationsTable featuresOff= "viewMenu"` when it is not selected. Nothing is written when it is selected.

3. **Show Format Menu:** Adds `fnd:applicationsTable featuresOff= "formatMenu"` when it is not selected. Nothing is written when it is selected.

4. **Icons and Text:** Set `fnd:applicationsTable defaultToolbarItemDisplay= "icon" or "iconAndText"`.

5. **Show Detach:** Adds `fnd:applicationsTable featuresOff= "detach"` when it is not selected. Nothing is written when it is selected.

6. **Show Wrap:** Adds `fnd:applicationsTable featuresOff= "wrap"` when it is not selected. Nothing is written when it is selected.

7. **Show Freeze:** Adds `fnd:applicationsTable featuresOff= "freeze"` when it is not selected. Nothing is written when it is selected.

8. **Show Status Bar:** Adds `fnd:applicationsTable featuresOff= "statusBar"` when it is not selected. Nothing is written when it is selected.

9. **Row Headers:**
   - If "False" does nothing.
   - If "True" sets `af:column rowHeader="true"` for every column defined by the view object as a key.
   - If "Unstyled (default)" sets `af:column rowHeader="unstyled"` for every column defined by the view object as a key.
Other actions:

- If nothing is selected in the CRUD patterns page, that is, nothing to go into the toolbars: set `fnd:applicationsTable primaryToolbarRendered = "false"` and `fnd:applicationsTable secondaryToolbarRendered = "false"

- Set `af:table styleClass = "verticalGridVisible"` if there are nested columns.

- If no CRUD patterns are set, leave `rowSelection=single` and remove the `selectedRowKeys` and `selectionListener` attributes. This prevents client server communications to synchronize the selected row.

Click Next to review the table structure.

**Figure 17-6  Review the Table Structure (Step 5 or 5)**
Click **Finish** to save your choices and create the Applications table, or click **Cancel** to delete your choices.

If you click **OK**, the table and its components appear in the editor.

**Figure 17-7  Applications Table and Its Components in the Editor**

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Enabled (Yes/No)</th>
<th>Zulu Offset</th>
<th>DST (Yes/No)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mezoneCode</td>
<td>#{_Name}</td>
<td>#{_EnabledFlag}</td>
<td>GMTSettled</td>
<td>DaylightSavingsFlag</td>
</tr>
<tr>
<td>mezoneCode</td>
<td>#{_Name}</td>
<td>#{_EnabledFlag}</td>
<td>GMTSettled</td>
<td>DaylightSavingsFlag</td>
</tr>
<tr>
<td>mezoneCode</td>
<td>#{_Name}</td>
<td>#{_EnabledFlag}</td>
<td>GMTSettled</td>
<td>DaylightSavingsFlag</td>
</tr>
</tbody>
</table>

17.1.2.2.1 Manually Enabling Delete Confirmation

This section describes how to enable Delete Confirmation, or add or edit the custom confirmation message, if you have an existing table.

When you set the `confirmDelete` attribute to `true`, the confirmation popup displays and the row is deleted when you click **Ok**. For this to work correctly, the `partialTriggers` on the `af:table` inside the `fnd:applicationsTable` should include `::confirm`, and the `::delete` and `::deleteConfirm` ids must be removed so the partialRefresh happens only when you click **Ok** in the popup. Setting `deleteImmediate="true"` when enabling delete confirmation sets the immediate attribute of the **Ok** button in the confirmation popup to true. The **Cancel** button of the delete confirmation popup has immediate set to true by default.

See **Example 17-2** for sample code that shows both the delete confirmation enabled and the custom message.

With `af:table` selected in the Structure view, **Figure 17-8** shows the PartialTriggers entries in the Property Inspector view.

**Figure 17-8  Delete Confirmation PartialTriggers in Property Inspector**

With `fnd:applicationsTable` selected in the Structure view in JDeveloper, the Property Inspector showing the custom message settings will appear similar to **Figure 17-9**.
**Figure 17-9  Message Settings in Property Inspector**

![Message Settings in Property Inspector](image)

**Example 17-2  Sample Code Showing Delete Confirmation and Custom Message**

```xml
<fnd:applicationsTable tableId="ATt2" id="AT2" confirmDelete="true"
    deleteMsg="#{viewcontrollerBundle.ARE_YOU_SURE_YOU_WANT_TO_DELETE}"
    deleteEnabled="true" createPatternType="inline"
    duplicatePatternType="inline" editPatternType="inline"
    exportEnabled="true"
    createText="#{viewcontrollerBundle.NEW}">
   <f:facet name="additionalToolbarButtons"/>
   <f:facet name="additionalActionItems"/>
   <f:facet name="appsTableSecondaryToolbar"/>
   <f:facet name="appsTableStatusbar"/>
   <f:facet name="appsTableViewMenu"/>
   <f:facet name="table">
      <af:table var="row" rowBandingInterval="0" id="ATt2"
        partialTriggers="::confirm ::create ::createMenuItem ::duplicate ::duplicateMenuItem">
   </f:facet>
</fnd:applicationsTable>
```

### 17.1.2.2.2 Multiple Row Selection on Table

If multiple row selection is enabled, editing functions will behave as shown in Table 17-3.

**Table 17-3  Function Behavior with Multiple Row Selection Enabled**

<table>
<thead>
<tr>
<th>Function</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete</td>
<td>Selecting more than one row and selecting Delete deletes all the selected rows.</td>
</tr>
<tr>
<td>Create</td>
<td>Selecting more than one row and selecting Create will create a new row as the first row.</td>
</tr>
<tr>
<td>Edit</td>
<td>Selecting more than one row and selecting Edit will show an alert window asking you to select a single row to edit.</td>
</tr>
<tr>
<td>Duplicate</td>
<td>Selecting more than one row and selecting Duplicate will show an alert window asking you to select a single row to duplicate.</td>
</tr>
</tbody>
</table>
Enabling Multiple Row Selection Manually

If multiple row selection is selected in the Create Applications Table wizard (see Figure 17-1), this step is not required.

You can not change pages created with single row selection to multiple row selection by just changing the `rowselection` attribute on the ADF table inside the Applications Table. This is because multiple row selection does not work with the `selectedrowkeys` attribute. To enable multiple row selection on existing tables, set `rowselection="multiple"` and remove the `selectedrowkeys` attribute, as shown in Example 17-3.

**Example 17-3   Example of Enabling Multiple Row Selection**

```xml
<fnd:applicationstable tableid="att3" id="at3" deleteenabled="true"
    createpattern="inline"
    dupplicatepattern="inline"
    editpattern="inline"
    createtext="#{viewcontrollerbundle.new}"
    <f:facet name="additionaltoolbarbuttons"/>
    <f:facet name="additionalactionitems"/>
    <f:facet name="table">
        <af:table value="#{bindings.gsflattable1.collectionModel}" selectionListener="#{bindings.gsflattable1.collectionModel.makeCurrent}" var="row" rows="#{bindings.gsflattable1.rangesize}"
            emptytext="#{bindings.gsflattable1.viewable ? applcorebundle.table_empty_text_no_rows_yet : applcorebundle.table_empty_text_access_denied}"
            fetchsize="#{bindings.gsflattable1.rangesize}"
            rowbandinginterval="0" id="att3"
            partialtriggers="::delete ::deletemenuitem ::create ::createmenuitem ::duplicate ::duplicatemenuitem ::selectionlistener ::selectedrowkeys"
            rowselection="multiple">
```

17.1.2.2.3 Toggle Click to Edit / Edit All in Applications Table

The Applications Table toolbar has an icon that can be clicked to toggle the Click to Edit and Edit All functions, and the View Menu on the toolbar includes the same toggle feature. Figure 17-10 shows the Edit All menu option and icon when the table is in the Click to Edit mode.

**Figure 17-10   Table Edit All Menu Option and Icon**

![Table Edit All Menu Option and Icon](image)

Figure 17-11 shows the Click to Edit menu option and icon when the table is in the Edit All mode.
The toggle mode should only display if the table is editable. If it contains only output components, there should be no toggle button. This is a true/false property (see toggleEditRendered in Table 17-2) on the Applications Table and does not happen automatically.

17.1.3 Introduction to Selected Elements in the Table Property Inspector

When a table is added to a page, code similar to that shown in Example 17-4 is inserted and displayed in the Source view.

Many of these settings are easily changed using the Table Property Inspector, which contains these sub-sections: Common, Patterns, Style, Customization, and Other. This section discusses certain selected settings.

Example 17-4  Sample Code Added When a Table Is Added

```
<fnd:applicationsTable tableId="AT1" id="AT1" confirmDelete="true"
    deleteMsg="#{viewcontrollerBundle.DO_YOU_WANT_TO_DELETE_THIS_ROW}"
    deleteEnabled="true" createPatternType="inline"
    duplicatePatternType="secondaryWindow"
    editPatternType="inline" exportEnabled="true"
    createText="#{viewcontrollerBundle.NEW}"
    duplicatePopUpId="Afp2">
    <f:facet name="additionalToolbarButtons"/>
    <f:facet name="additionalActionItems"/>
    <f:facet name="table">
        <af:table value="#{bindings.ServiceRequests1.collectionModel}"
                var="row" rows="#{bindings.ServiceRequests1.rangeSize}"
                emptyText="#{bindings.ServiceRequests1.viewable ?
                    applcoreBundle.TABLE_EMPTY_TEXT_NO_ROWS_YET :
                applcoreBundle.TABLE_EMPTY_TEXT_ACCESS_DENIED}"
                fetchSize="#{bindings.ServiceRequests1.rangeSize}"
                rowBandInt="0"
                selectedRowKeys="#{bindings.ServiceRequests1.collectionModel.selectedRow}"
                selectionListener="#{bindings.ServiceRequests1.collectionModel.makeCurrent}"
                rowSelection="single" id="AT1"
                partialTriggers="::confirm ::create ::createMenuItem">
```

17.1.3.1 Common Properties Section

The Common properties section of the Table Property Inspector resembles Figure 17-12.
The selected Common settings are:

- **Primary Toolbar Rendered**: Set this to False if no default actions or buttons will be used. If this is set to True and there will be no actions or buttons, the separators around buttons will display even if no button displays.

- **Secondary Toolbar Rendered**: Set this to False if no default actions or buttons will be used. If this is set to True and there will be no actions or buttons, the separators around buttons will display even if no button displays.

- **Actions Menu Rendered**: If no default actions were selected in the wizard, set this to False to avoid doubled separator lines.

- **ActionsContentDelivery**: Sets the Content Delivery attribute on the actions menu of the table. The options are Immediate (the default) and Lazy. Immediate populates the action menus as soon as the page is displayed. Lazy only populates an action menu when it is selected. There will be a slight delay the first time the menu is selected; there will be no delay the next time the menu is selected because the menu items are cached. You should set the ActionsContentDelivery to Lazy when you do not have any partialTriggers set on the items in the Actions menu because setting the value to Immediate affects the performance.

### 17.1.3.2 Patterns Properties

The Patterns properties section of the Table Property Inspector resembles Figure 17-13. The properties for Create, Duplicate, Edit, and Delete are the same.
The selected Patterns settings are:

- **Disabled**: Sets whether the button is disabled (shown as grayed). This does not determine if the button is displayed; it only sets its appearance and functionality.

- **Enabled**: Sets the rendered attribute on the Create/Duplicate/Edit/Delete button icon and menu item. If you are using the default action, a string called `create` (or `duplicate/edit/delete`) is returned.

- **Immediate**: Sets whether data validation - client-side or server-side - should take place when events are generated by the button. When immediate is set to `true`, the default ActionListener provided by the JavaServer Faces implementation should be executed during the Apply Request Values phase of the request processing lifecycle, rather than waiting until the Invoke Application phase.

- **Partial Triggers**: A partial trigger affects only the selected item, rather than the entire page. For instance, Example 17-5 sets the partialTrigger attribute value on the Create button icon and Create menu item.

**Example 17-5  Example of a Partial Trigger**

```xml
<af:inputComboboxListOfValues id="ledgerIdId" popupTitle="Search and Select: #{bindings.LedgerId.hints.label}" value="#{bindings.LedgerId.inputValue}" label="#{bindings.LedgerId.hints.label}" model="#{bindings.LedgerId.listOfValuesModel}" required="#{bindings.LedgerId.hints.mandatory}" columns="#{bindings.LedgerId.hints.displayWidth}" shortDesc="#{bindings.LedgerId.hints.tooltip}"
    <f:validator binding="#{bindings.LedgerId.validator}"/>
    <af:convertNumber groupingUsed="false" pattern="#{bindings.LedgerId.format}"/>
</af:inputComboboxListOfValues>

<fnd:applicationsTable tableId="AT1" id="AT1" deleteEnabled="true" createPatternType="inline"
    duplicatePatternType="inline" editPatternType="inline"
    createText="#{viewcontrollerBundle.NEW}"/>```
17.1.3.3 Other Properties

The Patterns properties section of the Table Property Inspector resembles Figure 17-14.

Figure 17-14 Other Properties Section

- AttachImmediate and AttachText: Do not use. These have been deprecated.

17.1.4 How to Modify Applications Table Components and Properties

When you create an Applications table in the Applications Table wizard, you can add data controls to the table and icons and menu actions to the table menu bar.

17.1.4.1 Add Data Controls to Tables

To add Data Controls to tables:

1. Find the data source in the Data Controls panel.

2. Drag and drop either the entire data source or individual fields:

   - To the table in the page Design view.

     Use the context menu that is displayed when you drag to the Design view to choose which component to use for this attribute.

     For example, you might drag and drop the data control component TimezoneServiceAMDataControl > Timezone > Name, then choose Create > Texts > ADF Input Text w/ Label from the context menu, as shown in Figure 17-15.

     Figure 17-15 Example of Context Menu Choices in Design View

   - To the page Structure view.
For example, to add a field from a data source to a table, drag the field from the data source to this path: \texttt{fnd:applicationsTable > f:facet – table > af:table <tableId>}. When you drop the field on the component, you are prompted to choose which component to use for this attribute. Using the example in Figure 17-16, you would choose either the ADF Read-only Column, or the ADF Column, depending on whether the fields need to be read-only or not.

\textbf{Figure 17-16} \hspace{1em} \textit{Example of Context Menu Choices in Structure View}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{image168x420.png}
\caption{Example of Context Menu Choices in Structure View}
\end{figure}

\subsection{17.1.4.2 Working with Table Menus and Icons}

To add buttons, icons and menu items to the table menu bar, in the Component Palette, drag and drop any component (such as an icon component) to the menuBar facet to add the component.

If you have multiple buttons added to the additionalToolbarButton facet, they may display vertically, instead of horizontally, at runtime. To correct this display, surround the toolbar buttons with an af:toolbar, as shown in Example 17-6:

\begin{example}
\textbf{Example 17-6 \hspace{1em} Surrounding the Toolbar Buttons with af:toolbar}

\begin{verbatim}
<af:facet name="additionalToolbarButtons">
  <af:toolbar>
    <af:commandToolbarButton text="Button1"/>
    <af:commandToolbarButton text="Button2"/>
  </af:toolbar>
</af:facet>
\end{verbatim}
\end{example}

\begin{note}
The Format menu is part of Applications Table. It provides several functions, including move rows and sort selection. Rows have to be selectable to enable this.

For tables and treeTables with selectable columns, the default top level menu items are View and Format. To turn off the Format menu, the af:table should not have selectable columns.
\end{note}
17.1.4.3 Increasing Table Width to Fill 100% of Its Container

Applications Table can be stretched by placing it in the center facet of an ADF panelStretchLayout component. Do not set the width using the inlineStyle attribute on either Applications Table or panelStretchLayout. For more information about basic page layout and the inlineStyle attribute, see the "Organizing Content on Web Pages" and the "Customizing the Appearance Using Styles and Skins" chapters in the Developing Web User Interfaces with Oracle ADF Faces.

17.1.4.4 Using an Applications Table with a Query Component

Note:

If you use just the ApplicationTable ID in the resultComponentId of the ADF Query component, the underlying table is not refreshed with the results of the search.

When using an Applications Table as a resultant table that shows the results from a search on a query component, follow these steps to set the resultComponentId attribute on af:query:

1. In the JSF page that contains the query component and the Applications Table, select the query component.

2. In the Property Inspector, select the resultComponentId property and then Edit.

3. From the edit panel, select the af:table (the ADF table that is present in the "table" facet of the Applications Table).

The resultComponentId would follow this format:

::<applicationsTableId>:_ATp:<tableId>

and would appear similar to:

resultComponentId="::AT1:_ATp:ATt7"

17.2 Implementing the Applications Tree

The Applications Tree component provides the following basic capabilities that satisfy the requirements specified in the user experience designs:

- Tree toolbar with default buttons
- Facets for adding ADF tree, custom toolbar buttons, and so on
- Default implementations for tree actions

17.2.1 How to Add an Applications Tree to Your Page

You can add an Applications Tree to your page in two ways.

- You can select the Applications Tree from the Applications component palette and drag and drop it on your page.
- You can drag and drop a data collection from the data control palette to your page and select the Applications Tree from the list of available UI components.
The facets shown in Table 17-4 are exposed on the Applications Tree.

### Table 17-4  Applications Tree Facets

<table>
<thead>
<tr>
<th>Facet</th>
<th>Description</th>
<th>Allowed Children</th>
</tr>
</thead>
<tbody>
<tr>
<td>tree</td>
<td>Facet for holding the ADF tree</td>
<td>ADF Tree</td>
</tr>
<tr>
<td>additionalToolbarButtons</td>
<td>Facet for adding toolbar button icons by the developer.</td>
<td>ADF Command Toolbar Buttons</td>
</tr>
<tr>
<td>additionalActionItems</td>
<td>Facet for adding more menu items to default menu items.</td>
<td>ADF menu item component</td>
</tr>
<tr>
<td>appsTreeSecondaryToolbar</td>
<td>Facet for adding more commandToolbar components to secondary toolbar.</td>
<td>ADF Command Toolbar Button component</td>
</tr>
<tr>
<td>appsTreeStatusBar</td>
<td>Facet for adding component containing statusbar item(s). These statusbar items are merged with standard items provided by the panelCollection.</td>
<td>ADF component</td>
</tr>
<tr>
<td>appsTreeViewMenu</td>
<td>Facet for adding Menu Item(s) to added to the default view menu of the panelCollection. To add multiple menuItems into the view menu please add af:group component containing af:menuItems.</td>
<td>ADF menu item component</td>
</tr>
<tr>
<td>appsTreeAfterToolBar</td>
<td>Facet for adding more commandToolbar button components to after toolbar. In this facet any toolbar buttons added appear in a separate row below the normal group of toolbars.</td>
<td>&quot;af:toolbar&quot; or &quot;af:groups&quot; of &quot;af:toolbars&quot;</td>
</tr>
<tr>
<td>popup</td>
<td>Facet for adding popups. See Using the Custom Wizard with Applications Popups.</td>
<td>Any number of popups under a layout component</td>
</tr>
</tbody>
</table>

**Important:** When a popup is used to create or duplicate a row in an Applications Tree, you need to write your own logic behind the popup’s Cancel button (Action/ActionListener) to remove the newly-created row.

This can be done by either:

- A managed bean method that removes the newly-created row.
- Setting the Cancel button’s Action property to the rollback method defined in the pageDef file. This method would be defined in the pageDef file if the “rollback” from the operations of the dataControl is dragged and dropped onto the page.

The properties shown in Table 17-5 are exposed on the Applications Tree.
### Table 17-5 Exposed Applications Tree Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Allowed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>The unique ID for this Applications Tree</td>
<td>string</td>
</tr>
<tr>
<td>rendered</td>
<td>Whether the Applications Tree is rendered or not</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>treeId</td>
<td>The unique ID of the ADF tree underneath this Applications Tree</td>
<td>string</td>
</tr>
<tr>
<td>createPatternType</td>
<td>Whether any Create pattern is enabled, and if yes, which pattern</td>
<td>none, secondaryWindow (then Create Popup Id must also be set), page</td>
</tr>
<tr>
<td>editPatternType</td>
<td>Whether any Edit pattern is enabled, and if yes, which pattern</td>
<td>none, secondaryWindow (then Edit Popup Id must also be set), page</td>
</tr>
<tr>
<td>duplicatePatternType</td>
<td>Whether any Duplicate pattern is enabled, and if yes, which pattern</td>
<td>none, inline (see &quot;Inline Duplicate Pattern Type&quot;), secondaryWindow (then Duplicate Popup Id must also be set), page</td>
</tr>
<tr>
<td>createEnabled</td>
<td>Rendered attribute for create</td>
<td>string</td>
</tr>
<tr>
<td>editEnabled</td>
<td>Rendered attribute for edit</td>
<td>boolean value or Expression Language expression</td>
</tr>
<tr>
<td>duplicateEnabled</td>
<td>Rendered attribute for duplicate</td>
<td>boolean value or Expression Language expression</td>
</tr>
<tr>
<td>deleteEnabled</td>
<td>Rendered attribute for delete</td>
<td>boolean value or Expression Language expression</td>
</tr>
<tr>
<td>createAction</td>
<td>Action binding for the Create icon</td>
<td>method expression</td>
</tr>
<tr>
<td>editAction</td>
<td>Action binding for the Edit icon</td>
<td>method expression</td>
</tr>
<tr>
<td>duplicateAction</td>
<td>Action binding for the Duplicate icon</td>
<td>method expression</td>
</tr>
<tr>
<td>deleteAction</td>
<td>Action binding for the Delete icon</td>
<td>method expression</td>
</tr>
<tr>
<td>createActionListener</td>
<td>Action listener binding for the Create icon</td>
<td>method expression</td>
</tr>
<tr>
<td>editActionListener</td>
<td>Action listener binding for the Edit icon</td>
<td>method expression</td>
</tr>
</tbody>
</table>

**createActionListener**
- If defined, this property can be used to supplement the default action specified by the Pattern Type property or completely override it.

**editActionListener**
- If defined, this property can be used to supplement the default action specified by the Pattern Type property or completely override it.
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Allowed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>duplicateActionListener</td>
<td>Action listener binding for the Duplicate icon</td>
<td>method expression</td>
</tr>
<tr>
<td></td>
<td>If defined, this property can be used to supplement the default action specified by the Pattern Type property or completely override it.</td>
<td></td>
</tr>
<tr>
<td>deleteActionListener</td>
<td>Action listener binding for the Delete icon</td>
<td>method expression</td>
</tr>
<tr>
<td></td>
<td>If defined, this property can be used to supplement or override the default action.</td>
<td></td>
</tr>
<tr>
<td>createPopupId</td>
<td>ID of the popup to be invoked when Create button is clicked</td>
<td>string</td>
</tr>
<tr>
<td>editPopupId</td>
<td>ID of the popup to be invoked when Edit button is clicked</td>
<td>string</td>
</tr>
<tr>
<td>duplicatePopupId</td>
<td>ID of the popup to be invoked when Duplicate button is clicked</td>
<td>string</td>
</tr>
<tr>
<td>createText</td>
<td>Overrides default label for Create menu item. This value will also be shown as the short description for the Create button.</td>
<td>expression</td>
</tr>
<tr>
<td>editText</td>
<td>Overrides default label for Edit menu item. This value will also be shown as the short description for the Edit button.</td>
<td>expression</td>
</tr>
<tr>
<td>duplicateText</td>
<td>Overrides default label for Duplicate menu item. This value will also be shown as the short description for the Duplicate button.</td>
<td>expression</td>
</tr>
<tr>
<td>deleteText</td>
<td>Overrides default label for Delete menu item. This value will also be shown as the short description for the Delete button.</td>
<td>expression</td>
</tr>
<tr>
<td>exportEnabled</td>
<td>Whether export is enabled</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>featuresOff</td>
<td>A list of default features to turn off for the panelCollection, such as detach (see featuresOff attribute of panelCollection for more details)</td>
<td>string</td>
</tr>
<tr>
<td>inlineStyle</td>
<td>The CSS styles to use for the panelCollection component inside the Applications Tree component. This is intended for basic style changes.</td>
<td>string</td>
</tr>
<tr>
<td>Note: Do not set the width using the inlineStyle attribute on either Applications Tree or panelStretchLayout. Applications Tree can be stretched by placing it in the center facet of an ADF panelStretchLayout component.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>styleClass</td>
<td>styleClass to use for the panelCollection component inside Applications Tree component.</td>
<td>string</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Allowed Values</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>createImmediate</td>
<td>Sets immediate attribute value of &quot;Create&quot; toolbar button and &quot;Create&quot; menu item.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>deleteImmediate</td>
<td>Sets immediate attribute value of &quot;Delete&quot; toolbar button and &quot;Delete&quot; menu item.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>duplicateImmediate</td>
<td>Sets immediate attribute value of &quot;Duplicate&quot; toolbar button and &quot;Duplicate&quot; menu item.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>editImmediate</td>
<td>Sets immediate attribute value of &quot;Edit&quot; toolbar button and &quot;Edit&quot; menu item.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>actionsMenuRendered</td>
<td>Sets rendered attribute value of the Actions menu. When CRUD actions are not turned on, and no af:commandMenuFormItem is added to the additionalActionItems facet, then actionsMenuRendered should be set to false so that an empty Actions menu would not be displayed.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>primaryToolbarRendered</td>
<td>Sets rendered attribute value of the primary toolbar. When CRUD actions, attach, export are not turned on, and no af:commandToolbarButton is added to additionalToolbarButtons facet, then primaryToolbarRendered should be set to false so that an empty toolbar would not be displayed.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>secondaryToolbarRendered</td>
<td>Sets rendered attribute value of the secondary toolbar. When no af:commandToolbarButton is added to appsTableSecondaryToolbar facet, then secondaryToolbarRendered should be set to false so that an empty toolbar would not be displayed.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>createDisabled</td>
<td>Disabled attribute for create</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>editDisabled</td>
<td>Disabled attribute for edit</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>duplicateDisabled</td>
<td>Disabled attribute for duplicate</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>deleteDisabled</td>
<td>Disabled attribute for delete</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>confirmDelete</td>
<td>Set this value if you want delete confirmation to come up. The default message is The selected record(s) will be deleted. Do you want to continue? To change this, use the deleteMsg attribute.</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>deleteMsg</td>
<td>Provide a customized delete confirmation message that can be shown in the popup.</td>
<td>String value or Expression Language expression</td>
</tr>
</tbody>
</table>
### Table 17-5  (Cont.) Exposed Applications Tree Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Allowed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>exportRowLimit</td>
<td>This property sets the row limit for exported rows when &quot;Export to Excel&quot; is used to export records to Excel. If this is not set, then the row limit is picked from the value that is set for the profile option FND_EXPORT_ROWLIMIT.</td>
<td>Positive integer.</td>
</tr>
</tbody>
</table>

### Inline Duplicate Pattern Type

For inline patterns, the ADF tree underneath the Applications Tree should get refreshed when the icon or the menu item is clicked. For this to happen, the ADF tree needs to know that it should partially refresh itself. For this, set the partialTriggers attribute on the ADF tree to the ids of the menu item and the icon. For example, to refresh the tree when the **Delete** menu item or icon is clicked, set partialTriggers="delete deleteMenuItem" on the ADF tree. The partialTriggers attribute is set by the Applications Tree Creation wizard automatically; Applications developers should not need to set it explicitly. **Example 17-7** shows a sample markup that is generated by the Applications Tree Creation wizard.

### Model

The Applications Tree does not expose any bindings to the model. However, components within the Applications Tree, like the ADF tree, will be bound to the model.

### Controller

The Applications Tree component ships a default managed bean that performs the following functions:

- Default event handlers for all toolbar button/menu item action events. Event handler delegates to custom action method if set on the button/menu item action property.
  - A new row is created in the data collection, a popup invoked with the newly created row available for inserting values (the UI for the popup to show input fields for the new row has to be created separately by the developer), when Create Pattern Type is secondaryWindow. After the popup is dismissed, the tree is refreshed to display the newly-created node:
    - If **No Node** is selected when the **Create** button or menu item is clicked: The new node is created in the first-level of the Tree.
    - If **Leaf Node** or **Expanded Parent Node** is selected when the **Create** button or menu item is clicked: The new node is created as a child of the selected node, and placed directly below it.
    - If **Collapsed Parent Node** is selected: The parent is expanded to show the newly created child node placed directly below it.
  - A new row is *not* added when the Create Pattern Type is **page**, and the developer is responsible for wiring the navigation to the page when the icon or menu item is clicked. Only a standard outcome is returned from the default handler. The developer could use this default outcome to define a navigation rule.
− The selected row is made available for editing in a popup when the Edit icon is clicked, and Edit Pattern Type is secondaryWindow.

− When the Edit icon is clicked, and Edit Pattern Type is page, only a standard outcome is returned.

− When the Duplicate icon or menu item is clicked: A new node is added into the tree when the Create icon is clicked, and Create Pattern Type is inline. All non-primary key values of the selected node are copied to the new node.

− If Duplicate Pattern Type is inline, the newly created node is placed next to the selected node.

− If Duplicate Pattern Type is popup, a popup invoked with the newly created row is available for modifying the duplicated values (the UI for the popup to show input fields for the new row has to be created separately by the developer).

− Clicking the Delete icon deletes the selected node. It currently does not perform a cascade delete when a parent node is selected for delete. Applications developers need to handle deleting the child nodes if it is necessary.

• If the secondaryWindow option is chosen for any pattern type, and the corresponding popup id is set for that button (mandatory), then selecting the button invokes the popup.

• If page is chosen for any pattern type, then a standard outcome is returned on clicking the button. Standard outcomes are: create, edit, duplicate and delete for the four respective toolbar buttons.

• A default selection listener for the ADF tree is provided (the markup shows selectionListener="#{ApplicationsTreeBean.treeSelectionHandler}"). If you need to add custom logic to the selection listener, you should call this default selection listener from the custom logic. The treeSelectionHandler method of the ApplicationsTreeBean provides the following behavior:

  − When xxxxPatternType="secondaryWindow" and when there is no popup configured for the level where the node must be created, the icon and the menu item are disabled by default. But this behavior can be overridden by the xxxxDisabled attribute, where “xxxx” could be create, edit or duplicate.

  − Calls the ADF default tree listener:
    #{bindings.xxxx.treeModel.makeCurrent}

Example 17-8 shows sample code for calling the default selection listener from the custom selection listener.

To allow developers access to some of the implementation, the Applications Tree exposes a public class, oracle.apps.fnd.appcore.patterns.ApplicationsTreeEventHandler, that contains default event handlers for all the buttons. The button methods are named as process<buttonName>, such as processCreate and processEdit. Applications developers writing custom action handlers can also use the default implementation by calling these methods.
Use

For example, to attach a custom button handler to the Create button, follow these steps.

1. Define a managed bean class, as shown in Example 17-9:

2. Register the managed bean in the faces-config of the project.

3. Select the Property Inspector for the Applications Tree, and choose the Create Action property. Set #{CustomEventHandler.processCreate} as the expression for the property.

Example 17-7  Sample Markup Generated by the Applications Tree Creation Wizard

```xml
<fnd:applicationsTree treeId="tree1" id="appsTree1"
  createPatternType="secondaryWindow"
  createPopupId="create1,create2"
  duplicatePatternType="inline"
  deleteEnabled="true">
  <af:tree value="#{bindings.ServiceRequestsView1.treeModel}"
    var="node" rowSelection="single"
    selectionListener="#{ApplicationsTreeBean.treeSelectionHandler}"
    id="tree1"
    partialTriggers="::duplicate ::duplicateMenuItem ::delete ::deleteMenuItem">
    ...
  </af:tree>
</fnd:applicationsTree>
```

Example 17-8  Calling the Default Selection Listener from the Custom Selection Listener

```java
String defaultListener = "#{ApplicationsTreeBean.treeSelectionHandler}";
FacesContext fc = FacesContext.getCurrentInstance();
ExpressionFactory ef = fc.getApplication().getExpressionFactory();
MethodExpression me = ef.createMethodExpression(fc.getELContext(), defaultListener,
  String.class, new Class[] {SelectionEvent.class});
me.invoke(fc.getELContext(), new Object[] {selectionEvent});
```

Example 17-9  Defining Managed Bean Class to Attach Custom Handler to a Button

```java
import oracle.apps.fnd.applcore.patterns.ui.ApplicationsTreeEventHandler;
import oracle.apps.fnd.applcore.patterns.ui.util.PatternUtils;

public class CustomEventHandler
{
  public String processCreate()
  {
    // Custom code
    ...

    // Call default event handler if required. It will return a standard outcome for
    // this button click.
    ApplicationsTreeEventHandler appTreeHandler =
    ApplicationsTreeEventHandler.getInstance();
    String outcome = appTreeHandler.processCreate();

    // If popup is required to be invoked after event handling
    PatternUtils.invokePopup(popupId);

    return outcome;
  }
}
```
17.2.1.1 Adding the Applications Tree

The Applications Tree can be added to a page or page fragment using either the Component First or the Data First approach. Valid drop locations in the page or page fragment include ADF Form, and ADF Layout components and the Applications Panel (jsp:root, af:form, af:root, fnd:applicationsPanel, af:group, af:panelBorderLayout, af:panelBox, af:panelCollection, af:panelFormLayout, af:panelGroupLayout, af:panelHeader, af:panelStretchLayout, af:showDetailItem, af:panelWindow, af:popup, af:showDetail, af:subform, f:facet, f:panelGrid, f:panelGroup, af:pageTemplateDef, af:pageTemplate#$<localArea_Facet>).

The Applications Tree can be added to a page or page fragment using either the Component First or the Data First approach. Both approaches launch a wizard that helps you quickly define the appropriate tree layout which adhere to the user experience standards. When you complete this wizard, you can further refine the tree definition by editing the resulting tree component as needed.

Component First

Navigate to the Component Palette. Click the list of libraries and select Applications. Drag the Applications Tree from the list of components and drop it onto the page. The wizard will launch after dropping the Applications Tree on the page.

Data First

Navigate to the Data Controls panel of the Application Navigator. Open the panel by clicking its bar, then navigate through the hierarchy to locate the data source that you would like to include in the Applications Tree. Select that data source and drag it onto the page. A context menu will appear with a list of components. Move the mouse over the Applications category list. Select Tree under the Applications menu to launch the Applications Tree wizard, as shown in Figure 17-17.

Figure 17-17  Data First Method

17.2.1.2 Applications Tree Create Wizard

The Applications Tree Create wizard consists of two panels: Create Applications Tree and Configure Tree Patterns.
Create Applications Tree Panel

The Create Applications Tree panel will vary depending on the approach used to launch the Applications Tree creation process.

Using the Data First approach the Bind Data Now properties are hidden. The selected data source is automatically bound to the tree.

With the Component First approach, it is up to the developer to decide whether to bind a Data Collection to the tree component. You can skip the data control binding step when creating the Applications Tree. In this case, the Applications Tree will create an ADF tree without data binding.

If you wish to bind a data control to the tree component using the Component First approach, check the Bind Data Now checkbox. This will enable the Browse button for the Data Source property. Click Browse to display a list of data sources available for binding. Navigate through the list, select the desired data source, and click OK, as shown in Figure 17-18.

Figure 17-18 Create Applications Tree Data Source

When the Data Source is selected, you can configure the ADF tree. Use the Add icon to add one of the children of the selected Data Source to be the next level of the tree, as shown in Figure 17-19.
Tree Level Rules: This pane displays rules that control the display order of the hierarchical tree or tree table UI components. The tree binding populates the tree UI component starting from the top of the Tree Level Rules list and continues until it reaches the last rule or until it encounters a rule whose accessor cannot find a target attribute. The more rules you choose, the more nodes you can display in the tree or tree table UI component.

Folder Label: Specify an Expression Language expression that selects labels to display in the tree, such as #{label.countryLabel}.
You also can use the Expression Language expression #{node.accessorLabel} to obtain information that allows you to traverse up and down a tree of data, not necessarily starting at the logical root node of the tree. This is useful if you want to access a parent node rather than the root node of the tree.

Enable Filtering: Select to filter the data that displays in the tree or tree table. After you select this checkbox, you can select an attribute on the data collection that will be used to filter the table data that display in the tree or tree table.

Available Attributes and Selected Attributes: The shuttle at the bottom of the Create Applications Tree panel allows you to control the attributes at each tree level you wish to display as a tree node in the tree. When finished, click Continue to proceed to the Configure Tree Patterns Dialog. Select Cancel to abort the creation of the Applications Tree.
Configure Tree Patterns Panel

Use the Configure Tree Patterns panel to select the default actions offered by your Applications Tree. See Figure 17-20.

**Figure 17-20  Configure Tree Patterns**

You may select any or all of the following five actions for your Applications Tree: Create, Duplicate, Edit, Delete and Export. If you enable Create, Duplicate, or Edit, you must choose the appropriate pattern that will be used to invoke that action (Inline, Popup, or Page).

- **Inline** - Perform the action on the current table row (only available for the Duplicate action)
- **Popup** - Bring up a div modal window on top of the current page for the requested action
- **Page** - Replace the current page or page fragment with a completely separate page or page fragment to perform the action. Page fragments are used when using bounded task flows.

The **Add** button for configuring the **Popup** button is enabled when the Popup pattern is selected. When you click **Add**, a dropdown of the data collection name of each tree level is displayed. You need to choose the tree level that needs the popup to be configured. When a data collection name is selected, the Applications Popup wizard is displayed. (See Using the Custom Wizard with Applications Popups.) This same data collection will automatically be bound to the Applications Popup. The Popup will also be defaulted as having Editable Content on the Window Buttons page in the wizard.

**Export**: Export the data to a Microsoft Excel-compatible file.

**Delete**: Allows users to delete the row.
• **Confirm Delete**: Select this option so that the default *The selected record(s) will be deleted. Do you want to continue?* prompt displays in a popup when the delete row function is used.

When you set the `confirmDelete` attribute to **true**, the confirmation popup displays and the row is deleted when you click **Ok**. For this to work correctly, the `partialTriggers` on the `af:tree` inside the `fnd:applicationsTree` should include `::confirm`, and the `::delete` and `::deleteConfirm` ids must be removed so the `partialRefresh` happens only when you click **Ok** in the popup. See [Manually Enabling Delete Confirmation](#).

• **Confirmation Message**: If you want to replace the default confirmation message with a custom one, enter the string here. The string will be converted to a text resource and added to the default resource bundle.

If you already have a confirmation message defined in a resource bundle, click the ellipsis and choose from the list.

When finished, click **OK** to complete creation of the Applications Tree. Selecting **Cancel** will abort the creation of the Applications Tree.

### 17.2.1.3 Working with the Applications Tree

This section discusses modifying settings in the JDeveloper Property Inspector. For more information, see [Introduction to Selected Elements in the Table Property Inspector](#).

**Editing - Properties**

When you have created the Applications Tree, you can modify the property values by using the Property Inspector. You can select the Applications Tree in one of three ways:

- Select the component in the Design view of the page.
- Select the `<fnd:applicationsTree ...>` line in the Source view of the page.
- Select `fnd:applicationTree` from the hierarchy in the Structure View.

All the components created as part of the Applications Tree are editable using this same approach, as shown in [Figure 17-21](#).

**Figure 17-21 Tree Property Inspector**

![Tree Property Inspector](image_url)

**Adding a Data Source**

When you have created the Applications Tree, you can add data controls to the facets / content containers within that tree using the following steps:
1. Navigate to the Data Controls panel of the Application Navigator.

2. Open the hierarchy to find the data source.

3. Drag and drop either the entire data source or individual fields into the correct location on the page. The correct location is dependent on the component.

**Adding UI Content**

To achieve the final goals for a page design, you will likely need to add other components to the Applications Tree. Common facets are provided to help you achieve these goals. The facet names and use are documented in the Facet tree of the Component Structure and Functions. For example, your tree may require an additional action beyond the standard actions that are provided by the Applications Tree. You can navigate to the Component Palette and drag and drop a commandToolbarButton component on to the additionalToolbarButtons facet to add a new icon to the Tree toolbar.

**Increasing Tree Width to Fill 100% of Its Container**

An Applications Tree can be stretched by placing it in the center facet of an ADF panelStretchLayout component. Do not set the inlineStylewidth on panelStretchLayout. For more information about basic page layout and the inlineStyle attribute, see the "Organizing Content on Web Pages" and the "Customizing the Appearance Using Styles and Skins" chapters in the Developing Web User Interfaces with Oracle ADF Faces.

### 17.3 Implementing Applications Tree Tables

The Application Tree Table component implements the tree table portion of the user experience pattern.

The Application Tree Table component provides these basic capabilities to satisfy the requirements specified in the user experience designs:

- Tree Table toolbar with default buttons.
- Facets for adding items such as ADF tree table and custom toolbar buttons.
- Default implementations for tree actions.

**Adding an Application Tree Table to Your Page**

You can add the Application Tree Table to your page in two ways.

- Select the Application Tree Table from the Applications component palette and drag and drop it on your page.
- Drag and drop a data collection from the data control palette to your page and select the Applications Tree Table from the list of available UI components.

The properties shown in Table 17-6 are exposed on the Applications Tree Table:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Allowed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Unique ID for this Applications Tree Table.</td>
<td>string</td>
</tr>
</tbody>
</table>
### Table 17-6  (Cont.) Applications Tree Table Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Allowed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>rendered</td>
<td>Whether the Applications Tree Table is rendered or not.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>treeTableId</td>
<td>Unique ID of the ADF tree table underneath this Applications Tree Table.</td>
<td>string</td>
</tr>
<tr>
<td>createPatternType</td>
<td>Whether any Create pattern is enabled, and if yes, which pattern.</td>
<td>none, inline, secondaryWindow (then Create Popup Id must also be set), page</td>
</tr>
<tr>
<td>editPatternType</td>
<td>Whether any Edit pattern is enabled, and if yes, which pattern.</td>
<td>none, inline, secondaryWindow (then Edit Popup Id must also be set), page</td>
</tr>
<tr>
<td>duplicatePatternType</td>
<td>Whether any Duplicate pattern is enabled, and if yes, which pattern.</td>
<td>none, inline (see Note), secondaryWindow (then Duplicate Popup Id must also be set), page</td>
</tr>
<tr>
<td>deleteMsg</td>
<td>Provide a customized delete confirmation message that can be shown in the popup.</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>deleteEnabled</td>
<td>Whether any Delete pattern is enabled.</td>
<td>boolean</td>
</tr>
<tr>
<td>createEnabled</td>
<td>Whether any Create pattern is enabled.</td>
<td>boolean</td>
</tr>
<tr>
<td>duplicateEnabled</td>
<td>Whether any Duplicate pattern is enabled.</td>
<td>boolean</td>
</tr>
<tr>
<td>editEnabled</td>
<td>Whether any Edit pattern is enabled.</td>
<td>boolean</td>
</tr>
<tr>
<td>confirmDelete</td>
<td>Set this value if you want delete confirmation to come up.</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>createAction</td>
<td>Action binding for the Create icon.</td>
<td>method expression</td>
</tr>
<tr>
<td>editAction</td>
<td>Action binding for the Edit icon.</td>
<td>method expression</td>
</tr>
<tr>
<td>duplicateAction</td>
<td>Action binding for the Duplicate icon.</td>
<td>method expression</td>
</tr>
<tr>
<td>deleteAction</td>
<td>Action binding for the Delete icon.</td>
<td>method expression</td>
</tr>
<tr>
<td>createActionListener</td>
<td>Action listener binding for the Create icon.</td>
<td>method expression</td>
</tr>
<tr>
<td>editActionListener</td>
<td>Action listener binding for the Edit icon.</td>
<td>method expression</td>
</tr>
<tr>
<td>duplicateActionListener</td>
<td>Action listener binding for the Duplicate icon.</td>
<td>method expression</td>
</tr>
<tr>
<td>deleteActionListener</td>
<td>Action listener binding for the Delete icon.</td>
<td>method expression</td>
</tr>
</tbody>
</table>
### Table 17-6 (Cont.) Applications Tree Table Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Allowed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>createPopupId</td>
<td>ID of the popup to be invoked when the Create button is clicked.</td>
<td>string</td>
</tr>
<tr>
<td>editPopupId</td>
<td>ID of the popup to be invoked when the Edit button is clicked.</td>
<td>string</td>
</tr>
<tr>
<td>duplicatePopupId</td>
<td>ID of the popup to be invoked when the Duplicate button is clicked.</td>
<td>string</td>
</tr>
<tr>
<td>deletePopupId</td>
<td>ID of the popup to be invoked when the Delete button is clicked.</td>
<td>string</td>
</tr>
<tr>
<td>createText</td>
<td>Overrides the default label for the Create menu item.</td>
<td>expression</td>
</tr>
<tr>
<td>editText</td>
<td>Overrides the default label for Edit menu item.</td>
<td>expression</td>
</tr>
<tr>
<td>duplicateText</td>
<td>Overrides the default label for Duplicate menu item.</td>
<td>expression</td>
</tr>
<tr>
<td>deleteText</td>
<td>Overrides the default label for Delete menu item.</td>
<td>expression</td>
</tr>
<tr>
<td>exportEnabled</td>
<td>Whether export is enabled.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>featuresOff</td>
<td>A list of default features to turn off for the panelCollection, for instance detach (see featuresOff attribute of panelCollection for more details)</td>
<td>string</td>
</tr>
<tr>
<td>inlineStyle</td>
<td>The CSS styles to use for the panelCollection component inside the Applications Tree Table component. This is intended for basic style changes.</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> Do not set the width using the inlineStyle attribute on either Applications Tree Table or panelStretchLayout. Applications Tree Table can be stretched by placing it in the center facet of an ADF panelStretchLayout component.</td>
<td></td>
</tr>
<tr>
<td>styleClass</td>
<td>styleClass to use for the panelCollection component inside the Applications Tree Table component.</td>
<td>string</td>
</tr>
<tr>
<td>createImmediate</td>
<td>Sets immediate attribute value of Create toolbar button and Create menu item.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>deleteImmediate</td>
<td>Sets immediate attribute value of Delete toolbar button and Delete menu item.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Allowed Values</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>duplicateImmediate</td>
<td>Sets immediate attribute value of Duplicate toolbar button and Duplicate menu item.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>editImmediate</td>
<td>Sets immediate attribute value of Delete toolbar button and Delete menu item.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>createDisabled</td>
<td>Disabled attribute for Create.</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>deleteDisabled</td>
<td>Disabled attribute for Delete.</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>editDisabled</td>
<td>Disabled attribute for Edit.</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>duplicateDisabled</td>
<td>Disabled attribute for Duplicate.</td>
<td>Boolean value or Expression Language expression</td>
</tr>
<tr>
<td>actionsMenuRendered</td>
<td>Sets the rendered attribute value of the Actions menu. When CRUD actions are not turned on, and no af:commandMenuItems is added to the additionalActionItems facet, actionsMenuRendered should be set to False so that an empty Actions menu will not be displayed.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>primaryToolbarRendered</td>
<td>Sets the rendered attribute value of the primary toolbar. When CRUD actions, attach, export are not turned on, and no af:commandToolbarButton is added to the additionalToolbarButtons facet, primaryToolbarRendered should be set to False so that an empty toolbar will not be displayed.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>secondaryToolbarRendered</td>
<td>Sets the rendered attribute value of the secondary toolbar. When no af:commandToolbarButton is added to the appsTableSecondaryToolbar facet, secondaryToolbarRendered should be set to False so that an empty toolbar will not be displayed.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>toggleEditRendered</td>
<td>The toggleEditRendered feature is used to render the editAll or clickToEdit choices for Applications Tree Table. See Toggle Click to Edit / Edit All in Applications Tree Table.</td>
<td>Boolean. Default value is False.</td>
</tr>
</tbody>
</table>
Table 17-6 (Cont.) Applications Tree Table Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Allowed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>PartialTriggers</td>
<td>Partial triggers attribute for the toolbar button.</td>
<td>String of IDs.</td>
</tr>
<tr>
<td></td>
<td>For example: deletePartialTriggers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The partial triggers property of the Create, Edit, Duplicate and Delete buttons, and menu items are exposed. Users can enable and disable buttons according to rows selected or other actions carried out on the page. The same partialTrigger attribute for each one is used both for the command toolbar button and the menu item. For example, when the createPartialTriggers is set in the Applications Tree Table, the value for this attribute is set on the partialTrigger property of both the create command toolbar button and create menu item.</td>
<td></td>
</tr>
</tbody>
</table>

**Important:** The PartialTriggers attribute must be entered manually by the developer. This is because, at design time, the JDeveloper Property Inspector can:
- Select the incorrect ID.
- Append square brackets around the selected id, such as [id1 id2].

**Example 1:**
To disable the Edit, Delete and Duplicate buttons when the table is empty, set this property on the editDisabled, deleteDisabled or duplicateDisabled property of the Applications Tree Table.

\[#{bindings.VOiterator.estimatedRowCount == 0 ? true : false}\]
where VOiterator is your iterator name

**Example 2:**
Disable any of the buttons in the Applications Tree Table according to the functional rules or by setting disable=false when create is selected on an empty table (considering these buttons were disabled following Example 1).

To do this, create an attribute binding on the view object attribute that will decide whether the row can be deleted/edited/duplicated. For example, you can use a binding similar to this example on the disable property of a button:

\[#{bindings.MyAttrBinding.inputValue == 'compare value' ? true : false}\]

Add Partial Page Refresh (PPR) on the button to the table ID of the af:table. This does not require any change in the selectionListener of the table. Keep the default one.
Table 17-6  (Cont.) Applications Tree Table Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Allowed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>exportRowLimit</td>
<td>This property sets the row limit for exported rows when &quot;Export to Excel&quot; is used to export records to Excel. If this is not set, then the row limit is picked from the value that is set for the profile option FND_EXPORT_ROWLIMIT.</td>
<td>Positive integer.</td>
</tr>
<tr>
<td>defaultToolbarItemDisplay</td>
<td>Use to hide the DETACH/FREEZE/WRAP labels and just display the related icons in the toolbar to reclaim space used by the display of the labels on the table’s panel collection.</td>
<td>iconAndText will display the labels.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>icon will hide the labels.</td>
</tr>
</tbody>
</table>

Note:

For inline patterns, the ADF tree table beneath the Applications Tree Table should be refreshed when the icon or the menu item is clicked. For this to happen, the ADF tree table needs to know that it should partially refresh itself. To do this, set the partialTriggers attribute on the ADF tree table to the Ids of the menu item and icon. For example, to refresh the tree table when the Delete menu item is selected or the icon is clicked, set partialTriggers="delete deleteMenuItem" on the ADF tree table. The partialTriggers attribute is set by the Applications Tree Table Creation wizard automatically. Applications developers should not need to set it explicitly. Example 17-10 shows a sample markup that is generated by the Applications Tree Table Creation wizard.

Table 17-7 shows the facets that are exposed on the Applications Tree Table.

Table 17-7  Applications Tree Table Facets

<table>
<thead>
<tr>
<th>Facet</th>
<th>Description</th>
<th>Allowed Children</th>
</tr>
</thead>
<tbody>
<tr>
<td>treeTable</td>
<td>Facet for holding the ADF tree table.</td>
<td>ADF Tree Table</td>
</tr>
<tr>
<td>additionalToolbarButtons</td>
<td>Facet for adding toolbar icons by the developer.</td>
<td>ADF Command Toolbar Buttons</td>
</tr>
<tr>
<td>additionalActionItems</td>
<td>Facet for adding more menu items to default menu items.</td>
<td>ADF menu item component</td>
</tr>
<tr>
<td>appsTreeTableAfterToolbar</td>
<td>Facet for adding an af:toolbar or af:groups of af:toolbars that appear in a separate row below the normal group of toolbars.</td>
<td>ADF Toolbar or ADF Groups of Toolbar</td>
</tr>
<tr>
<td>appsTreeTableSecondaryToolBar</td>
<td>Facet for adding more commandToolbar components to the secondary toolbar.</td>
<td>ADF Command Toolbar Button component</td>
</tr>
</tbody>
</table>
### Table 17-7 (Cont.) Applications Tree Table Facets

<table>
<thead>
<tr>
<th>Facet</th>
<th>Description</th>
<th>Allowed Children</th>
</tr>
</thead>
<tbody>
<tr>
<td>appsTreeTable Statusbar</td>
<td>Facet for adding component containing statusbar items. These statusbar items are merged with standard items provided by the panelCollection.</td>
<td>ADF component</td>
</tr>
<tr>
<td>appsTreeTable ViewMenu</td>
<td>Facet for adding Menu Items to the default view menu of the panelCollection. To add multiple menuItems to the view menu, add an af:group component containing af:menuItems.</td>
<td>ADF menu item component</td>
</tr>
<tr>
<td>popup</td>
<td>Facet for adding popups. See Using the Custom Wizard with Applications Popups.</td>
<td>Any number of popups under a layout component.</td>
</tr>
</tbody>
</table>

**Important:** When a popup is used to create or duplicate a row in an Applications Tree Table, you need to write your own logic behind the popup’s Cancel button (Action/ActionListener) to remove the newly-created row.

This can be done by either:

- A managed bean method that removes the newly-created row.
- Setting the Cancel button’s Action property to the rollback method defined in the pageDef file. This method would be defined in the pageDef file if the "rollback" from the operations of the dataControl is dragged and dropped onto the page.

## Model

The Applications Tree Table does not expose any bindings to the model. However, components within the Applications Tree Table, such as the ADF tree table, will be bound to the model.

## Controller

The Applications Tree Table component ships a default managed bean (internal to the Oracle Fusion Middleware Extensions for Applications team) that performs the following functions that will only work with rowSelection="single" on the ADF tree table:

- Default event handlers for all toolbar button/menu item action events. Event handler delegates to custom action method if set on the button/menu item action property.

  - When Create Pattern Type is secondaryWindow, a new row is created in the data collection and a popup is invoked with the newly-created row available for inserting values. (The UI for the popup to show input fields for the new row has to be created separately by the developer.) After the popup is dismissed, the tree table is refreshed to display the newly-created row:

    * If no row is selected when the Create button or menu item is clicked, the new row is created in the first-level of the Tree.
* If Leaf Node or Expanded Parent Node is selected when the Create button/menu item is clicked, the new row is created as a child of the selected node and placed directly below it.

* If the Collapsed Parent Node is selected, the parent is expanded to show the newly-created child node that placed directly below it.

* A new row is not added when the Create Pattern Type is page. Only a standard outcome is returned

* A new row is added into the tree table when the Create icon is clicked, and Create Pattern Type is inline.

* The selected row is made available for editing in a popup when the Edit icon is clicked, and Edit Pattern Type is secondaryWindow.

* When the Edit icon is clicked and Edit Pattern Type is page, only a standard outcome is returned.

* When the Duplicate icon or menu item is clicked, a new node is added into the tree when the Create icon is clicked, and Create Pattern Type is inline. All non-primary key values of the selected node are copied to the new node.

* If Duplicate Pattern Type is inline, the newly-created row is placed above the selected row.

* If Duplicate Pattern Type is popup, a popup invoked with the newly-created row is available for modifying the duplicated values (the UI for the popup to show input fields for the new row has to be created separately by the developer).

* Clicking the Delete icon deletes the selected row. It currently does not perform cascade delete when a parent node is selected for delete. The Applications developer needs to handle deleting the child nodes if it is necessary.

• If the secondaryWindow option is chosen for any pattern type, and the corresponding popup id is set for that button (mandatory), selecting the button invokes the popup.

• If page is chosen for any pattern type, a standard outcome is returned on clicking the button. Standard outcomes are Create, Edit, Duplicate and Delete for the four respective toolbar buttons.

• The Oracle Fusion Middleware Extensions for Applications (Applications Core) provides a default selection listener for the ADF tree (the markup shows: selectionListener="#{ApplicationsTreeBean.treeSelectionHandler}"). If a developer needs to add custom logic to selection listener, the developer should call this default selection listener from the custom logic. The treeSelectionHandler method of ApplicationsTreeBean provides the following behavior:

  – When xxxxPatternType="secondaryWindow" and when there is no popup configured for the level where the node must be created, the icon and the menu item are disabled by default. But this behavior can be overridden by xxxxDisabled attribute ("xxxx" could be "create", "edit" or "duplicate").
- Calls the ADF default tree listener:
  #{bindings.xxxx.treeModel.makeCurrent} See Example 17-11.

To allow Applications developers access to some of the implementation, the
Applications Tree Table exposes a public class
oracle.apps.fnd.appcore.patterns.ApplicationsTreeEventHandler
that contains default event handlers for all the buttons. The button methods are named
as process<buttonName>, such as processCreate and processEdit. Developers writing
custom action handlers can also use the default implementation by calling these
methods.

Example
To attach a custom button handler to the Create button:

1. Define a managed bean class, as shown in Example 17-12.
2. Register the managed bean in the faces-config of the project.
3. Open the Property Inspector for the Applications Tree Table and choose the
Create Action property. Set #{CustomEventHandler.processCreate} as the
expression for the property.

Example 17-10  Sample Markup Generated by the Applications Tree Table Creation
Wizard

```
<fnd:applicationsTreeTable treeTableId="treeTable1" id="appsTree1"
createPatternType="secondaryWindow"
createPopupId="create1,create2"
duplicatePatternType="inline"
deleteEnabled="true">
  <f:facet name="treeTable">
    <af:treeTable value="#{bindings.ServiceRequestsView1.treeModel}"
var="node" rowSelection="single"
selectionListener="#{ApplicationsTreeBean.treeSelectionHandler}"
id="treeTable1"
partialTriggers="::duplicate ::duplicateMenuItem ::delete ::deleteMenuItem">
```

Example 17-11  Sample Code for Calling the Default Selection Listener from a
Custom Selection Listener

```
String defaultListener = "#{ApplicationsTreeBean.treeSelectionHandler}";
FacesContext fc = FacesContext.getCurrentInstance();
ExpressionFactory ef = fc.getApplication().getExpressionFactory();
MethodExpression me =
  ef.createMethodExpression(fc.getELContext(), defaultListener,
String.class, new Class[]{SelectionEvent.class});
me.invoke(fc.getELContext(), new Object[] {selectionEvent});
```

Example 17-12  Define a Managed Bean Class to Attach Custom Handler to a Button

```
import oracle.apps.fnd.appcore.patterns.ui.ApplicationsTreeEventHandler;
import oracle.apps.fnd.appcore.patterns.ui.util.PatternUtils;

public class CustomEventHandler
{
  public String processCreate()
  {
    // Custom code
    ...
```
// Call default event handler if required. It will return a standard outcome for this button click.
ApplicationsTreeEventHandler appTreeHandler = ApplicationsTreeEventHandler.getInstance();
String outcome = appTreeHandler.processCreate();

// If popup is required to be invoked after event handling
PatternUtils.invokePopup(popupId);

return outcome;
}

17.3.1 How to Add an Applications Tree Table

The Applications Tree Table can be added to a page or page fragment using either the Component First or the Data First approach. Both approaches launch a wizard that is intended to help you quickly define the appropriate tree layout that adheres to the user experience standards. When you complete this wizard, you can further refine the tree definition by editing the resulting tree component as needed.


Component First

Navigate to the Component Palette. Click the list of libraries and select Applications. Drag the Applications Tree Table from the list of components and drop it onto the page to launch the wizard.

Data First

Navigate to the Data Controls panel of the Application Navigator. Open the panel and navigate through the hierarchy to locate the data source that you would like to include in the Applications Tree Table. Select that data source and drag it to the page. A context menu will display a list of components. Select Tree under the Applications menu to launch the Applications Tree Table wizard, as shown in Figure 17-22.

Figure 17-22 Data First Method to Add a Tree Table
17.3.1.1 Applications Tree Table Create Wizard

The Applications Tree Table Create wizard consists of four panels: Create Applications Tree Table, Select Tree Table Columns, Configure Tree Table Patterns, and Summary.

Create Applications Tree Table Panel

This step creates a tree binding and node definitions of the tree. The Create Applications Tree Table Panel will vary depending on the approach used to launch the Applications Tree Table creation process.

Using the Data First approach, the Bind Data Now properties are hidden. The selected data source is automatically bound to the tree.

With the Component First approach, the developer must decide whether to bind a Data Collection to the tree table component. You can skip the data control binding step when creating the Applications Tree Table. In this case the Applications Tree Table will create an adf tree table without data binding.

If you wish to bind a data control to the tree component using the Component First approach, select the Bind Data Now checkbox. This will enable the Browse button for the Data Source property. Click Browse to display a list of data sources available for binding. Navigate through the list and select the desired data source. Click OK, as shown in Figure 17-23.

![Create Applications Tree Table Panel](image)

When the Data Source is selected, you can configure the ADF tree. Click the Add icon to add one of the children of the selected Data Source to be the next level of the tree, as shown in Figure 17-24.
The shuttle at the bottom of the Create Applications Tree Table panel allows you to select the attributes at each tree level you wish to display as a tree node or columns in the tree table, as shown in Figure 17-25.

When finished, click **Next** to proceed to the Select Tree Table Columns panel. Select **Cancel** to abort the creation of the Applications Tree Table.

**Select Tree Table Columns Panel**

The Select Tree Table Columns panel shown in Figure 17-26 allows you to select an attribute for displaying as node stamp and select an attribute for displaying as path stamp. You can also configure the columns to be displayed inside the tree table here. When finished, click **Next** to proceed to the Configure Tree Table Patterns Dialog. Selecting **Cancel** will abort the creation of the Applications Tree Table.
The Configure Tree Table Patterns panel allows you to select the default actions offered by your Applications Tree Table, shown in Figure 17-27.

Figure 17-27 Configure Tree Table Patterns
You may select any or all of the following five actions for your Applications Tree Table: Create, Duplicate, Edit, Delete and Export. If you enable Create, Duplicate, or Edit, you must choose the appropriate pattern that will be used to invoke that action (Inline, Popup, Page).

- **Inline** - Perform the action on the current table row.
- **Popup** - Bring up a div modal window on top of the current page for the requested action.
- **Page** - Replace the current page or page fragment with a completely separate page or page fragment to perform the action. (Page fragments are used when using bounded task flows.)

The **Add** button for configuring the Popup button is enabled when the Popup pattern is selected. When you click **Add**, a dropdown of the data collection name of each tree level is displayed. You need to choose the tree level that needs the popup to be configured. When a data collection name is selected, the Applications Popup Wizard is displayed. (See **Using the Custom Wizard with Applications Popups**.) This same data collection will automatically be bound to the Applications Popup. The Popup will also be defaulted as having Editable Content on the Window Buttons page in the wizard. Refer to **Using the Custom Wizard with Applications Popups**.

**Export**: Export the data to a Microsoft Excel-compatible file.

**Delete**: Allows users to delete the row.

- **Confirm Delete**: Select this option so that the default **The selected record(s) will be deleted. Do you want to continue?** prompt displays in a popup when the delete row function is used.

  When you set the **confirmDelete** attribute to true, the confirmation popup displays and the row is deleted when you click **Ok**. For this to work correctly, the **partialTriggers** on the **af:treetable** inside the **fnd:applicationsTreeTable** should include **::confirm**, and the **::delete** and **::deleteConfirm** ids must be removed so the **partialRefresh** happens only when you click **Ok** in the popup. See **Manually Enabling Delete Confirmation**.

- **Confirmation Message**: If you want to replace the default confirmation message with a custom one, enter the string here. The string will be converted to a text resource and added to the default resource bundle.

  If you already have a confirmation message defined in a resource bundle, click the ellipsis and choose from the list.

Click **Finish** to complete creation of the Applications Tree Table. Select **Cancel** to abort the creation of the Applications Tree Table.

**17.3.1.2 Working with the Applications Tree Table**

When you have created the Applications Tree Table, you can modify the property values by using the Property Inspector editor. The Property Inspector for the Applications Tree Table component can be viewed by selecting the component in the page. You can select the Applications Tree Table in one of three ways:

- Select the component in the Design view of the page.
- Select the `<fnd:applicationsTreeTable...>` line in the Source view of the page.
- Select `fnd:applicationTreeTable` from the hierarchy in the Structure View. All of the components created as part of the Applications Tree Table are editable using this same approach, as shown in Figure 17-28.

**Figure 17-28  Tree Table Property Inspector**

17.3.1.2.1 Add a Data Source

When you have created the Applications Tree Table, you can add data controls to the facets and content containers within that tree table using the following steps:

1. Navigate to the Data Controls panel of the Applications Navigator.
2. Expand the hierarchy to find the data source.
3. Drag and drop either the entire data source or individual fields into the correct location on the page. The correct location depends on the component.

17.3.1.2.2 Adding UI Content

To achieve the final goals for a page design, you probably will need to add other components to the Applications Tree Table. Common facets are provided to help you achieve these goals. The facet names and use are documented in the Facet table of the Component Structure and Functions. For example, the tree table may require additional actions beyond the standard actions that are provided by the Applications Tree Table. You can open the Component Palette and drag and drop a `commandToolbarButton` component onto the `additionalToolbarButtons` facet to add a new icon to the Tree Table toolbar.

17.3.1.2.3 Increasing Tree Table Width to Fill 100% of Its Container

An Applications Tree Table can be stretched by placing it in the center facet of an ADF `panelStretchLayout` component. Do not set the `inlineStylewidth` on `panelStretchLayout`. For more information about basic page layout and the `inlineStyle` attribute, see the "Organizing Content on Web Pages" and the

---

Implement UIs in JDeveloper with Application Tables, Trees and Tree Tables 17-55
"Customizing the Appearance Using Styles and Skins” chapters in the *Developing Web User Interfaces with Oracle ADF Faces*.

17.3.1.2.4 Toggle Click to Edit / Edit All in Applications Tree Table

The Applications Tree Table toolbar has an icon that can be clicked to toggle the Click to Edit and Edit All functions, and the View Menu on the toolbar includes the same toggle feature. This functions the same as described for the Applications Table in *Toggle Click to Edit / Edit All in Applications Table*.

17.4 Using the Custom Wizard with Applications Popups

**Note:**

`af:popup` is a generic function documented in the *Oracle Fusion Middleware Developing Web User Interfaces with Oracle ADF Faces* guide. This section discusses using the Popup wizard in JDeveloper Oracle Fusion design time.

---

Popups are an option when editing rows. While the standard `af:popup` component does not provide buttons or data binding, the Applications Popup wizard provides the base `af:popup` with:

- a title
- standard buttons
- customized button capability
- data binding
- code that developers can use to invoke the popup
- design-time support
- popup facets and properties that can be customized

Popups can be used as standalone components or with the patterns shown in Table 17-8.

**Table 17-8 Patterns That Require Popups**

<table>
<thead>
<tr>
<th>Pattern Set</th>
<th>Patterns</th>
<th>Description/User Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attachments</td>
<td>Attachments Field, Attachments Column</td>
<td>When users display attachments, the resulting popup displays the current attachment, and also allows users to add new attachments to an entity.</td>
</tr>
<tr>
<td>Compare Objects</td>
<td>Configure Comparison, One to Many, Configure and Compare</td>
<td>When users select objects to compare and click Compare, the resulting popup display allows users to choose comparison criteria and the objects to be compared. Users can change comparison criteria after each comparison is displayed.</td>
</tr>
</tbody>
</table>
Table 17-8  (Cont.) Patterns That Require Popups

<table>
<thead>
<tr>
<th>Pattern Set</th>
<th>Patterns</th>
<th>Description/User Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Create Multiple Objects</td>
<td>When users click Create Multiple Objects or choose from a menu or toolbar, the resulting popup allows users to create multiple objects.</td>
</tr>
<tr>
<td>Detail On Demand</td>
<td>Popup Details</td>
<td>When users select a table row and click an information button, the resulting popup displays information about the selected row.</td>
</tr>
<tr>
<td>Edit</td>
<td>Secondary Window Edit</td>
<td>When users select records and click Edit on a table toolbar, the resulting popup allows users to edit those records.</td>
</tr>
<tr>
<td>Information Entry Form</td>
<td>Secondary Window</td>
<td>When users click a button, the resulting popup allows users to enter data into the popup.</td>
</tr>
<tr>
<td>Record Navigation</td>
<td>Secondary Window Detail</td>
<td>When users navigate through data records, the resulting popup allows users to edit the records.</td>
</tr>
<tr>
<td>Transactional Search/Results</td>
<td>Popup Window</td>
<td>When users click Search within an application and enter search criteria, the popup displays the search results.</td>
</tr>
</tbody>
</table>

17.4.1 Creating a Popup

You create Applications Popups in the wizard that is displayed when you add the popups to your previously-created JavaServer Faces (JSF) pages (or page fragments) from the Data Controls panel. You also can create popups from within other applications component wizards, such as Applications Table.

17.4.1.1 How to Add Applications Popups to JSF Pages or Page Fragments

Before you can add popup components, you must add the Applications Popup to your pages from the Data Controls panel.

To add a Popup using a data control:

1. In the Application Navigator, open the Data Controls panel.
2. Navigate to the data source that you want to bind to the popup.
3. Drag and drop the data control to the JSF page, as shown in Figure 17-29.
4. In the Create context menu that is displayed, choose Applications > Popup.

The Popup wizard is displayed.

To add an applications popup from a table:

1. Launch the Applications Table wizard, such as by selecting the Applications option from the Component Palette and then clicking the Table option.

2. In the Configure Table Patterns dialog, set the Create, Duplicate or Edit Pattern to Popup.

3. Click Configure Popup to display the Popup wizard.

4. Follow the steps in How to Add Applications Popup Components Using the Wizard.

---

**Note:**

The data source for the table becomes the default data source for the popup.

---

**17.4.1.2 How to Add Applications Popup Components Using the Wizard**

This section explains how to use the Popup wizard to add components to your popups.

In the Popup wizard you can:

- Create titles and related layout.
- Add, move, or delete components.
- Define buttons.
- Create a preview.

All mandatory fields in the wizard contain default values, allowing you to accept the defaults and work through the steps quickly. Clicking Cancel on any of the dialogs cancels the popup creation and discards any values you entered.

Clicking Finish on any of the dialogs has the following effects:

- Displays a preview of the popup.
- Creates the popup with the values you provided on that screen and any previous screens, and default values for the remaining screens.
Caution:
Each wizard dialog contains a Messages field that displays errors for that step. Do not proceed to the next wizard step without correcting the errors in the present step.

Using the Applications Popup Wizard:
When the Popup wizard launches, the Set Title and Panel Layout dialog is displayed, as shown in Figure 17-30.

**Figure 17-30  Set Title and Panel Layout Dialog**

1. Choose title and panel properties:
   - Enter a popup Id. The Id is a string that must be unique to the page fragment. It is used when other components want to be related to this component. For example, if you have a table with Create in Popup, the Create button needs to include this popup Id so it can call it.
   - Enter a title for the popup window.
     The title is prepopulated with the Oracle Fusion Applications Standard for the title, which is a combination of the action of the task, the type of object, and the specific object name:

     \[
     \text{[Action] [Object Type]: [Object Name]}
     \]

     The title should be a reference to a single message with appropriate tokens, because, according to Oracle internationalization standards, you should not concatenate translatable messages in the code. See the "Internationalizing and Localizing Pages" chapter in the *Developing Web User Interfaces with Oracle ADF Faces*, and the expanded information in "To add an Applications Table using the Applications Table wizard:"

     So, in this example, the title reference in the JSF page fragment source would resemble:
but in the resource bundle, it would be defined as:

```xml
<trans-unit id="EDIT_INVOICE"> <source>Edit Invoice: {INVOICE_NUMBER}</source> <target/></trans-unit>
```

- [Optional] Enter a title icon location or click Browse to navigate to the icon's location.

- Select the number of layout columns -- 2 or 3 -- for the popup. This affects the data you bind to. For instance, if you are showing an address with name, street, and town, you could put this information in two or three columns.

2. Click Next to display the Select Components to Display dialog, as shown in Figure 17-31.

![Select Components to Display Dialog](image)

- The Components table is automatically populated with the data source fields.

- In the Fields section:
  - Click X to delete the selected component.
  - Click + to add a component.
  - Click the Reorder icons to change the position of the selected component in the list.
  - Click an entry in the Display Label column to enter a new label, such as Dept. Number instead of DeptNo.
  - Click an entry in the Component to Use column to choose a component from the list, as shown in Figure 17-32.
Figure 17-32  Components to Use List

See the description of "Components to Use" in the "Using Attributes to Create Text Fields" section in the Developing Fusion Web Applications with Oracle Application Development Framework.

- Click **Next** to display the Enable Window Buttons dialog, as shown in Figure 17-33.

**Figure 17-33  Enable Window Buttons Dialog**

3. In the Enable Window Buttons dialog, you can perform the following actions:

- If the popup requires navigation workflow, such as when choices within the popup lead users to another window:
  - Check the **Enable Record Navigation** box.
  - Choose **linear** or **non-linear** navigation from the **Navigation Type** menu.

- Choose **View Only Content** to create a read-only popup, or choose **Editable Content** to allow users to edit the popup.

When you choose View Only Content, you automatically enable Slots 1 and 2. When you choose Editable Content, you automatically enable Slots 1, 2, and 3, as shown in **Figure 17-34**.
Click **Next** to display the Review Panel Configuration dialog, as shown in Figure 17-35.

4. The dialog displays a list of your choices.

   In the dialog:
   - Click **Back** to revise your choices.
   - Click **Finish** to save your choices and create the popup.
   - Click **Cancel** to delete the popup information and exit the wizard.

5. When you click **Finish**, the `af:popup` component is displayed in the page, as shown in Figure 17-36.
17.4.2 How to Modify Popup Components and Properties

This section describes how to bind data sources to the popup, and edit popup components and properties after you create the popup in the Popup wizard.

17.4.2.1 Access the Popup on a JSF Page

Before you can edit the popup components and properties or bind the popup to a data source, you must access the popup in the Property Inspector section of your JSF page.

To Access the Popup on a JSF Page

1. Make the JSF page the active file.

2. Access the popup by double-clicking one of the following on the JSF page:
   - `af:popup` component in the Design view.
   - `<af:popup...>` line in the Source view.
   - `af:popup` entry in the Structure view hierarchy.

   When you select the popup, the Popup - Property Inspector is displayed below the page.

17.4.2.2 Add a Data Source to an Existing Popup

This section describes how to add a data source after you have created a popup in the Popup wizard.

To add a data source to an existing popup:

1. In the Application Navigator, expand the Data Control panel.

2. Navigate to the data source you want to bind to the `af:popup`.

3. Drag and drop the entire data source, or individual fields, to the JSF page in Design mode.

   Data-source fields are bound to popup components. Components are stored in the contents facet under the `af:panelFormLayout` hierarchy.

17.4.2.3 Add User-Interface Content to an Existing Popup

To add components to an existing popup, drop the new components into facets.
To add UI content to existing popups:

1. Open the Component Palette.
2. In the list, choose the ADF Faces library.
3. Drag and drop the component to the appropriate popup facet.
   For example, to add a new button, drag and drop the icon to the buttonBar facet.

17.4.2.4 Adding action and actionListener Methods to the Popup Buttons

Either method bindings or managed bean methods can be assigned to the action and actionListener attributes to provide functionality for the buttons that were selected for the popup at design time.
This chapter discusses the Applications Panels, Master-Detail, Hover and Dialog Details components used to implement user interface features in JDeveloper. This chapter includes the following sections:

- Implementing Applications Panels
- Implementing the Simple Panel
- Implementing Applications Master-Detail
- Implementing Hover
- Implementing Applications Dialog Details

For basic information, see:

- Getting Started with Your Web Interface
- Implementing the UI Shell
- Implementing UIs in JDeveloper with Application Tables, Trees and Tree Tables

### 18.1 Implementing Applications Panels

Applications panels help you create the following user interface (UI) components as part of the UI Applications patterns:

- Page title
- Form title
- Page button bar (including navigation bar)
- Facets for page-specific UI components

You must use Applications panels to standardize layout and appearance for all your page forms and buttons, including read-only pages.

**Before you begin:**

Before you can use Applications panels, you must be familiar with JDeveloper and be able to create JavaServer Faces (JSF) pages.

### 18.1.1 Overview of Applications Panel Components

Applications panels provide a button bar containing these buttons:
• **Canceling processes**: Cancel, Revert

• **Data-saving processes**: Save, Submit, Save and Continue, Save and Next, Save and Create Another, Continue, Create Another, Save and Close

• **Navigational processes**: Next, Previous, Back

The buttons are organized into four *slots*, as shown in Figure 18-7.

All panel buttons have attributes, and some buttons have facets. Button attributes include button qualities, such as the title string and the button name. Button facets are locations that contain panel data, such as content locations and button information locations.

Table 18-1 contains attributes that are exposed for the buttons.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Unique identification number for the panel.</td>
<td>string</td>
</tr>
<tr>
<td>rendered</td>
<td>Whether the panel is rendered (that is, converted from an object-based description into a graphical image for display).</td>
<td>boolean or expression</td>
</tr>
<tr>
<td>title</td>
<td>Panel title.</td>
<td>string or expression</td>
</tr>
<tr>
<td>navigationType</td>
<td>Type of navigation for that panel.</td>
<td>Navigation types:</td>
</tr>
<tr>
<td></td>
<td>• linear - sends users to an adjacent or contiguous window. This commonly is used when a series of actions or steps need to be followed in a sequential order.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• nonlinear - sends users to a non-adjacent or non-contiguous window. This is used when an action does not need to take place in a specific sequence.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• none - navigation is disabled.</td>
<td></td>
</tr>
<tr>
<td>&lt;button_name&gt;Visible</td>
<td>Whether the button is visible in the UI.</td>
<td>boolean or expression</td>
</tr>
<tr>
<td>For example:</td>
<td>submitVisible</td>
<td></td>
</tr>
<tr>
<td>&lt;button_name&gt;Rendered</td>
<td>Whether the button is rendered in the UI (that is, converted from an object-based description into a graphical image for display).</td>
<td>boolean or expression</td>
</tr>
<tr>
<td>For example:</td>
<td>submitRendered</td>
<td></td>
</tr>
<tr>
<td>&lt;button_name&gt;Action</td>
<td>Type of action that the button performs.</td>
<td>EL expression</td>
</tr>
<tr>
<td>&lt;button_name&gt;PopupId</td>
<td>ID of the popup that appears when users press the button.</td>
<td>string</td>
</tr>
<tr>
<td>For example:</td>
<td>submitPopupId</td>
<td></td>
</tr>
</tbody>
</table>
### Table 18-1 (Cont.) Attributes of Standard Panel Buttons

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;button_name&gt;ShortDesc</code></td>
<td>Tooltip text for the button.</td>
<td>string</td>
</tr>
<tr>
<td>For example: previousShortDesc</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>&lt;button_name&gt;Disabled</code></td>
<td>Next and Previous buttons only: Whether the button should be disabled in the UI.</td>
<td>boolean or expression</td>
</tr>
<tr>
<td>For example: submitDisabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>submitText</td>
<td>Submit button text: Text associated with the OK button.</td>
<td>string or expression</td>
</tr>
<tr>
<td>scrollable</td>
<td>Sets to true when scroll bar must be enabled. When scrollable is set to true, it sets <code>layout=&quot;scroll&quot;</code> on the af:panelGroupLayout component inside Applications Panel. Thus, the developer does not need to place af:panelGroupLayout with layout=&quot;scroll&quot; directly under the Applications panel.</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>saveOptionsStyle</td>
<td>Sets the appearance of the Save button. The Save button can be rendered as a normal button, or as a drop button, depending on the value of this attribute. When it is set to dropButton, the developer is expected to have other save options turned on (such as saveAndContinue or saveAndClose), or add af:commandMenuItem to the saveButtonMenu facet.</td>
<td>button or dropButton</td>
</tr>
<tr>
<td>savePartialSubmit</td>
<td>Sets the partialSubmit attribute on the Save button. For instance, you can set partialSubmit=true on the Save button to prevent a popup from being dismissed when the Save button is clicked.</td>
<td>True or false (default)</td>
</tr>
<tr>
<td>instructionText</td>
<td>The instructionText attribute places instruction text for the Applications Panel title. This instruction text appears right below the title if the collaborationToolbar facet and scalingInfo facet are empty. If they are not empty, the instructionText is placed under the scalingInfo that appears after the collaboration toolbar.</td>
<td>The instructionText attribute can take a String value or an EExpression. A helpTopicId can be passed to this attribute as an EExpression. For example: instructionText=&quot;#{adfFacesContext.helpProvider['helpTopicId'].instructions}&quot; or instructionText=&quot;#{adfFacesContext.helpProvider['helpTopicId'].definition}&quot; or instructionText=&quot;#{adfFacesContext.helpProvider['helpTopicId'].externalUrl}&quot;</td>
</tr>
</tbody>
</table>
### Table 18-1 (Cont.) Attributes of Standard Panel Buttons

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>panelToolbarRendered</td>
<td>If no default buttons that are provided by the Applications Panel are used, set this value to false to avoid displaying unnecessary separators.</td>
<td>Boolean</td>
</tr>
<tr>
<td>revertImmediate</td>
<td>Sets the <code>immediate</code> attribute on the Revert button. Sets whether data validation - client-side or server-side - should take place when events are generated by this component. When immediate is true, the default ActionListener provided by the JavaServer Faces implementation should be executed during the Apply Request Values phase of the request processing lifecycle, rather than waiting until the Invoke Application phase.</td>
<td>Boolean. Default is <code>false</code>.</td>
</tr>
<tr>
<td>submitStyle</td>
<td>Sets the appearance of the <strong>Submit</strong> button. The <strong>Submit</strong> button can be rendered as a normal button, or as a drop button, depending on the value of this attribute. When it is set to <code>dropButton</code>, the developer is expected to have other submit options turned on, or add <code>af:commandMenuitem</code> to the <code>submitButtonMenu</code> facet.</td>
<td>String. The two values are <code>button</code> (the default) and <code>dropButton</code>.</td>
</tr>
<tr>
<td>previousPartialSubmit</td>
<td>Sets the <code>partialSubmit</code> attribute on the <strong>Previous</strong> button.</td>
<td>True or false (default)</td>
</tr>
<tr>
<td>nextPartialSubmit</td>
<td>Sets the <code>partialSubmit</code> attribute on the <strong>Next</strong> button.</td>
<td>True or false (default)</td>
</tr>
<tr>
<td>saveAndCreateAnotherText</td>
<td>Sets the text that is displayed on the <strong>saveAndCreateAnother</strong> button.</td>
<td>String</td>
</tr>
<tr>
<td>createAnotherText</td>
<td>Sets the text that displays on the <strong>createAnother</strong> button.</td>
<td>String</td>
</tr>
<tr>
<td><code>&lt;button_name&gt;</code>Action</td>
<td>Sets the action attribute on the button with <code>&lt;button_name&gt;</code>. Users must provide their own action; there is no default action.</td>
<td>String or EL Expression.</td>
</tr>
<tr>
<td><code>&lt;button_name&gt;</code>ActionListener</td>
<td>Sets the actionListener attribute on the button with <code>&lt;button_name&gt;</code>. Users must provide their own actionListener; there is no default actionListener.</td>
<td>EL Expression.</td>
</tr>
<tr>
<td><code>&lt;button_name&gt;</code>PartialTriggers</td>
<td>partialTriggers attribute for <code>&lt;button_name&gt;</code> button.</td>
<td>String or EL Expression.</td>
</tr>
</tbody>
</table>

**Important:** The `PartialTriggers` attribute must be entered manually by the developer. This is because, at design time, the JDeveloper Property Inspector can:

- Select the incorrect ID.
- Append square brackets around the selected id, such as `[id1 id2]`. 

Implementing Applications Panels
Table 18-1  (Cont.) Attributes of Standard Panel Buttons

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>submitAccessKey</td>
<td>submitAccessKey property for the Submit toolbar button. submitAccessKey maps to submitText and accepts one character given to the submitText property as the access key. submitAccessKey sets the accessKey HTML property internally.</td>
<td>Character</td>
</tr>
<tr>
<td></td>
<td>The value to this must be one of the characters given for the submitText property to get an access key. The character will be displayed as underlined on the component name.</td>
<td></td>
</tr>
<tr>
<td>createAnotherAccessKey</td>
<td>createAnotherAccessKey property for the createAnother button. createAnotherAccessKey maps to createAnotherText and accepts one character given to the createAnotherText property as the access key. createAnotherAccessKey sets the accessKey HTML property internally.</td>
<td>Character</td>
</tr>
<tr>
<td></td>
<td>The value to this must be one of the characters given for the createAnotherText property to get an access key. The character will be displayed as underlined on the component name.</td>
<td></td>
</tr>
<tr>
<td>saveAndCreateAnotherAccessKey</td>
<td>saveAndCreateAnotherAccessKey property for the saveAndCreateAnother commandMenuItem. saveAndCreateAnotherAccessKey maps to saveAndCreateAnotherText and accepts one character given to the saveAndCreateAnotherText property as the access key. saveAndCreateAnotherAccessKey sets the accessKey HTML property internally.</td>
<td>Character</td>
</tr>
<tr>
<td></td>
<td>The value to this must be one of the characters given for the saveAndCreateAnotherText property to get an access key. The character will be displayed as underlined on the component name.</td>
<td></td>
</tr>
</tbody>
</table>

By default, a managed bean that ships with the Applications Panel enables certain actions when certain conditions exist. For example, default actions occur when users click buttons, and when developers set certain Applications Property values. These default actions are overridden if you change the value of the default button action property.

Table 18-2 contains facets that are exposed for each panel button.

Table 18-2  Facets of Standard Panel Buttons

<table>
<thead>
<tr>
<th>Facet</th>
<th>Description</th>
<th>Allowed Children</th>
</tr>
</thead>
<tbody>
<tr>
<td>contents</td>
<td>Facet for holding developer-defined content or content generated at design time.</td>
<td>Any ADF component.</td>
</tr>
<tr>
<td>navigationBar</td>
<td>Facet for holding the navigation choice list if the chosen Record Navigation Type is non-linear.</td>
<td>ADF selectOneChoice</td>
</tr>
<tr>
<td>actionButtonBar</td>
<td>Facet for holding custom action buttons.</td>
<td>ADF commandButtons and commandToolbarButtons under some ADF layout components.</td>
</tr>
<tr>
<td>saveButtonMenu</td>
<td>Facet for holding the custom menu and menu items for the Save button.</td>
<td>af:commandMenuItem or af:group</td>
</tr>
</tbody>
</table>
Table 18-2  (Cont.) Facets of Standard Panel Buttons

<table>
<thead>
<tr>
<th>Facet</th>
<th>Description</th>
<th>Allowed Children</th>
</tr>
</thead>
<tbody>
<tr>
<td>submitButtonMenu</td>
<td>Facet for holding the custom menu and menu items of the Submit button.</td>
<td>None</td>
</tr>
<tr>
<td>popup</td>
<td>Facet for holding any popups required for any of the buttons.</td>
<td>Applications popups under some ADF layout components.</td>
</tr>
<tr>
<td>appsPanelLegend</td>
<td>Facet for displaying legend information on the header.</td>
<td></td>
</tr>
<tr>
<td>appsPanelContext</td>
<td>Facet for displaying context information next to the header. The contextual information is displayed next to the header text.</td>
<td></td>
</tr>
<tr>
<td>customSaveDropButton</td>
<td>Facet for adding custom Save drop button.</td>
<td>This facet should contain <code>&lt;af:group&gt;</code> with <code>&lt;af:commandToolbarButton&gt;</code> under it. The Design Time handles this for you. See Figure 18-8 and its description.</td>
</tr>
<tr>
<td>localContext</td>
<td>Facet for adding content into local context region.</td>
<td></td>
</tr>
</tbody>
</table>
Table 18-2  (Cont.) Facets of Standard Panel Buttons

<table>
<thead>
<tr>
<th>Facet</th>
<th>Description</th>
<th>Allowed Children</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskStamp</td>
<td>Facet for adding a task stamp. There are three styles: one applied to the right side of the data, one to the left, and one to the container having these values. (AFStampContainer, TaskStampTextLabel, AFTaskStampTextValue) For every row of data in the taskStamp, a panelGroupLayout and two outputText components need to be added. Example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;af:panelGroupLayout layout=&quot;vertical&quot; valign=&quot;top&quot; styleClass=&quot;AFStampContainer&quot; id=&quot;ptpgl5&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;af:panelGroupLayout layout=&quot;horizontal&quot; halign=&quot;end&quot; id=&quot;ptpgl6&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;af:outputText value=&quot;Last Updated&quot; styleClass=&quot;TaskStampTextLabel&quot; id=&quot;ptot8&quot;/&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;af:outputText value=&quot;08-Nov-2007&quot; styleClass=&quot;AFTaskStampTextValue&quot; id=&quot;ptot9&quot;/&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/af:panelGroupLayout&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;af:panelGroupLayout layout=&quot;horizontal&quot; halign=&quot;right&quot; id=&quot;ptpgl7&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;af:outputText value=&quot;Budget Remaining&quot; styleClass=&quot;TaskStampTextLabel&quot; id=&quot;ptot10&quot;/&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;af:outputText value=&quot;$20,000.00&quot; styleClass=&quot;AFTaskStampTextValue&quot; id=&quot;ptot11&quot;/&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt;/af:panelGroupLayout&gt;</td>
<td></td>
</tr>
</tbody>
</table>
### Table 18-2  (Cont.) Facets of Standard Panel Buttons

<table>
<thead>
<tr>
<th>Facet</th>
<th>Description</th>
<th>Allowed Children</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>collaborationToolbar</td>
<td>Facet for adding collaboration toolbar buttons.</td>
<td></td>
<td>&lt;f:facet name=&quot;collaborationToolbar&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><a href="">af:toolbox</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><a href="">af:toolbar</a></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;af:commandImageLink text=&quot;One&quot; icon=&quot;/image1&quot; id=&quot;mycmd1&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;af:commandImageLink text=&quot;Two&quot; icon=&quot;/image2&quot; id=&quot;mycmd2&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;af:commandImageLink text=&quot;Three&quot; icon=&quot;/image3&quot; id=&quot;mycmd3&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;/af:toolbar&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;/af:toolbox&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;/f:facet&gt;</td>
</tr>
<tr>
<td>scalingInfo</td>
<td>Facet for adding scaling information.</td>
<td></td>
<td>&lt;af:panelGroupLayout layout=&quot;vertical&quot; styleClass=&quot;AFStampContainer&quot; id=&quot;pgl3&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;af:outputText value=&quot;AUD = Australian Dollar&quot; id=&quot;ot5&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Example for scalingInfo with more than one value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;af:panelGroupLayout layout=&quot;vertical&quot; styleClass=&quot;AFStampContainer&quot; id=&quot;pgl3&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;af:outputText value=&quot;AUD = Australian Dollar</td>
</tr>
<tr>
<td>submitButtonMenu</td>
<td>Facet for holding the custom menu and menu items for the Submit button.</td>
<td></td>
<td>af:commandMenuItem or af:group</td>
</tr>
<tr>
<td>Facet</td>
<td>Description</td>
<td>Allowed Children</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------</td>
<td></td>
</tr>
</tbody>
</table>
| contentsStretch     | The contents facet is a child of the panelGroupLayout so that scrolling can be enabled around the contents. But the panelGroupLayout does not allow its children to be stretched. The contentsStretch facet is not a child of the panelGroupLayout. Components placed inside this do not need to use an inlineStyle to set width and with declarative components placing them inside a panelStretchLayout will stretch the components. **Note:** The Applications Panel can be stretched by placing it in the center facet of an ADF panelStretchLayout component. Do not set the width using the inlineStyle attribute on either Applications Panel or panelStretchLayout. To use this facet, place your components inside the contentsStretch facet and set the attribute contentsFacet="stretch" on the Applications Panel. The user needs to trade between using scrollable or stretchable contents. A switcher reads the contentsFacet attribute from the ApplicationsPanel component to decide which facet to use. The default facet is scroll; to use the contentsStretch facet, set contentsFacet="Stretch" on the Applications Panel. Example:  
```xml
<fnd:applicationsPanel id="AP1" title="#{viewControllerBundle.APPLICATIONS_PANEL__STRETCH_FA}"
  scrollable="true"
  navigationType="none"
  cancelVisible="true"
  cancelRendered="true"
  submitVisible="true"
  submitRendered="true"
  contentsFacet="stretch">
  <f:facet name="contentsStretch">
    <af:panelStretchLayout id="psl1">
      <f:facet name="bottom"/>
      <f:facet name="center">
        <fnd:applicationsTable tableId="ATt2" id="AT2"
          deleteEnabled="true"
          createPatternType="inline"
          duplicatePatternType="inline"
          editPatternType="inline"
``` | Scroll (the default) or Stretch. |
Table 18-2 (Cont.) Facets of Standard Panel Buttons

<table>
<thead>
<tr>
<th>Facet</th>
<th>Description</th>
<th>Allowed Children</th>
</tr>
</thead>
<tbody>
<tr>
<td>createText=</td>
<td>&quot;#{viewControllerBundle.NEW}&quot;</td>
<td></td>
</tr>
<tr>
<td>appsPanelTrain</td>
<td>Facet for adding a horizontal train above header.</td>
<td></td>
</tr>
<tr>
<td>contentsFacet</td>
<td>This can be either scroll (the default) or stretch. Set to stretch when using the contentsStretch facet.</td>
<td></td>
</tr>
</tbody>
</table>

Model

The Applications panel does not expose any bindings to the model. However, components within the panel can be bound to the model.

Controller

The Applications Panel component ships with a default managed bean (internal to the Oracle Fusion Middleware Extensions for Applications team) that performs the following functions:

- Default event handlers for all button action events. Event handler delegates to custom action method if set on the button action property. Each button event handler simply returns a standard outcome which is the name of the button clicked. For example, clicking the Cancel button will return an outcome “cancel”.

- If popup ID is set for any button, selecting the button invokes the popup.

To allow developers access to some of the implementation, the Applications Panel exposes a public class `oracle.apps.fnd.appcore.patterns.ApplicationsPanelEventHandler` that contains default event handlers for all the buttons. The button methods are named as `process<buttonName>` such as `processSave` and `processCancel`. Application developers writing custom action handlers can also use the default implementation by calling these methods.

18.1.1.1 Implement a Custom Button Handler

Follow these steps to attach a custom button handler to the Cancel button.

1. Define the managed bean class, as shown in Example 18-1.

2. Register the managed bean in your project’s `faces-config` file.

3. Open the Property Inspector for the Applications panel and choose the Cancel Action property. As shown in the example in Figure 18-1, set `#{CustomeEventHandler.processCancel}` as the expression for the property.
4. Click the **Create** tab.

5. In the **Create Action** expression field, enter the following expression:

   `#{CustomEventHandler.processCreate}`

6. Click **Set**.

**Example 18-1    Example of Attaching a Custom Handler to a Button**

```java
import oracle.apps.fnd.applcore.patterns.ui.ApplicationsPanelEventHandler;
import oracle.apps.fnd.applcore.patterns.ui.util.PatternUtils;

public class CustomEventHandler {
    public String processCancel() {
        // Custom code
        ...
        // Call default event handler if required. It will return a standard outcome
        // for this button click.
        ApplicationsPanelEventHandler appPanelEventHandler = ApplicationsPanelEventHandler.getInstance();
        String outcome = appPanelEventHandler.processCancel();

        // If popup is required to be invoked after event handling
        PatternUtils.invokePopup(popupId);

        return outcome;
    }
}
```

### 18.1.2 How to Create an Applications Panel

You create Applications panels in the Applications Panel wizard, which is displayed when you add panels to your JSF pages (or page fragments) from the Component Palette or the Data Controls panel.

To Add an Applications Panel Using the Component Palette:

1. Open the **Component Palette**.

2. In the list, choose **Applications**.

3. In the list, click **Panel**. JDeveloper will attempt to place the panel at the current cursor location. If the current location is not appropriate, an error message
displays. You also can drag the Panel icon to the page in either the Design or the Source view. A plus (+) sign will be added to the arrow when it is over a location in which a panel can be inserted.

The Applications Panel wizard is displayed.

To Add an Applications Panel Using the Data Control Dialog:

1. In the Application Navigator, open the **Data Control** panel.
2. Navigate to the data source that you want to bind to the Applications panel.
3. Drag and drop the data control to the JSF page.
4. In the Create context menu that is displayed, choose **Applications > Panel**.

The Applications Panel wizard is displayed.

### 18.1.2.1 Add Applications Panels Using the Applications Panel Wizard

This section explains how to use the Applications Panel wizard to add panels to your page.

In the Applications Panel wizard you can:

- Specify panel titles and subsections
- Select panel components
- Bind a data source to the panel
- Arrange panel components
- Select panel buttons

In any Applications Panel wizard dialog, click **Cancel** to cancel your actions and exit the wizard. Click **Finish** on any dialog to accept the defaults and exit the wizard.

**To Add an Applications Table Using the Applications Panel Wizard:**

When the Applications Panel wizard is launched, the Title and Subsections dialog is displayed, as shown in Figure 18-2.
1. In the dialog:

- Enter the panel title. In the example, LABEL should be predefined in a bundle as "Edit Journal: ID {OBJ_ID}".

The title is prepopulated with the Oracle Fusion Applications Standard for the title, which is a combination of the action of the task, the type of object, and the specific object name:

[Action] [Object Type]: [Object Name]

The Object Name usually is something specific so you can identify a specific object. For instance, if you were dealing with part numbers, the Object Name could be a specific part number; if you were dealing with customer information, it could be the customer's name.

The title should be a reference to a single message with appropriate tokens, because, according to Oracle internationalization standards, you should not concatenate translatable messages in the code. See the Internationalizing and Localizing Pages section in the Oracle Fusion Middleware Developing Web User Interfaces with Oracle ADF Faces. You should use the following message formatters: af:formatNamed/af:formatFor example:

- Click the Add icon (+) to add Panel Subsections, or click the Delete icon (X) to delete the highlighted subsection.

Each subsection has editable title fields, panel type fields (Panel Header for a basic view or Show Detail Header for a more detailed view), and number of columns (1-3) fields.

The Panel Subsections is used to divide the Applications Panel facet (contents) with other layout components, such as panelHeader, show detail header, and panelGroupLayout. This lets the developer decide the layout during Design Time without needing to add each of these layouts manually after the panel creation. Of course, the user can add more or new layouts as needed after the panel is created.
Use the up or down arrows to change row order.

2. **Click Next.**

The Select an initial set of panel components dialog is displayed, as shown in Figure 18-3.

![Figure 18-3  Selecting an Initial Set of Panel Components](image)

3. In the Panel Components dialog:

   a. **Click Read-only Form** to create a read-only form. (optional) If you select Read-only, the choices in the Component to Use column will change from Input Text to Output Text types.

   b. **If** you have added the panel from the Component Palette, the Bind Data Now field displays. To bind a data source to the panel component:

      • **Select Bind Data Now.**
      
      • **Click Browse** to display the Data Source dialog, shown in Figure 18-4.
Figure 18-4    Data Source Dialog

Select the data source, then click **OK** to add it to the component. Optionally, you can bind the component to a data source at a later time.

When you choose a data source, the component fields in the dialog are automatically populated with the data source fields, which contain panel-component information.

c. To reorder component fields, click the up and down arrows. To delete component fields, click the Delete icon (X). You will be able to add more component fields later. See Adding a Data Source to an Existing Panel.

d. **Display Label:** In general, the labels defined in the selected Data Control will be what you want and you can leave this value at the default <Default> setting. Otherwise, enter a new label name.

e. **Value Binding:** In general, the label and the Value Binding will match and you can accept the displayed value. Otherwise, you can click in the field to display a drop-down list of the values available in the selected Data Control.

f. **Component To Use:** Data in Dialog Details can be read-only or updatable. Component to Use is similar to what Component does while creating a table. Clicking it reveals a choice list of values, and the dialog details popup would then at runtime show that particular column from the datacontrol as the selected component to use. The choice list is changed according to whether you choose read-only. If you selected Read-only, the choices will change from Input Text to Output Text types.

4. Click **Next.** The Components Layout dialog is displayed, as shown in Figure 18-5.
5. Drag components from their default locations to your previously defined subsections. The result will appear similar to Figure 18-6.

Figure 18-6 Example of Component Layout

6. Click Next. The Page Buttons dialog is displayed, as shown in Figure 18-7.
7. In the Page Level Buttons dialog:

- To enable panel navigation (optional):
  - Select **Enable Navigation**.
  - Choose a navigation type (Linear or Non-Linear).
    - Linear sends users to an adjacent or contiguous window. This commonly is used when a series of actions or steps need to be followed in a sequential order.
    - Non-Linear sends users to a non-adjacent or non-contiguous window. This is used when an action does not need to take place in a specific sequence.

- To disable editing of panel contents, select **View Only Content**.

- To enable editing of panel contents:
  - Select **Editable Content**.
  - Choose the transactional buttons to display in each panel slot from the respective slot dropdown menus.

  Note that Slot 3 defaults to Continue. However, as shown in Figure 18-7, if you select Submit, a text input field displays to the right. You can enter alternate text that makes more sense in your application for the submit action, such as OK or Purchase.

  You can create a Save or Submit pull down menu. When you choose Save in Slot 2, or Save and Close in Slot 3, an Add Menu option will appear. Click it to display a list similar to Figure 18-8.
These are options that can appear in a pull down menu at runtime under Save. To select, click the option you want. To add more than one, select Add Menu again and choose a second option. As they are chosen, check marks will appear next to each selected item, shown in Figure 18-9.

When an item is selected from the Add Menu of Slot 3, the selection of the drop-down in Slot 3 will become the label of af:commandToolbarButton and the selections in the Add Menu will become the af:commandMenuItem under af:menu in the popup facet of the af:commandToolbarButton. The af:commandToolbarButton will be added to the customSaveDropButton facet (see Table 18-2).

If an option is chosen in the Add Menu of Slot 2, it will be grayed-out as an option for Slot 3 to prevent you from making the same choice multiple times, as shown in Figure 18-10.

• **Enable Social Network**

The Enable Social Network and Social Row fields become available if the view object that was selected as the Data Source (in step 2 of 5) is Oracle Social Network enabled. You can deselect the option and edit the Social Row value if necessary. See Implementing Oracle Social Network Integration.  

8. Click Next. The Summary dialog is displayed, as shown in Figure 18-11.
9. Check to make sure your panel choices are correct.

10. Click Finish to create the panel. When you run this page, it will appear similar to Figure 18-12.

---

18.1.3 How to Modify Applications Panels Components and Properties

This section describes how to edit Applications Panel properties and components, how to add a data source to the panel, and how to add more UI content.

18.1.3.1 Stretching the Applications Panel

The Applications Panel can be stretched by placing it in the center facet of an ADF panelStretchLayout component. Do not set the width using the inlineStyle attribute on either Applications Panel or panelStretchLayout. For more
information about basic page layout and the inlineStyle attribute, see the Organizing Content on Web Pages and Customizing the Appearance Using Styles and Skins sections in the Oracle Fusion Middleware Developing Web User Interfaces with Oracle ADF Faces guide.

18.1.3.2 Access the Applications Panel on a JSF Page

Before you can edit the panel properties and components or bind the panel to a data source, you must access the panel in the Property Inspector section of your JSF page.

To access the panel on a JSF page:

1. Make the JSF page the active file.
2. Access the panel by double-clicking one of the following on the JSF page:
   - Applications Panel component in the Design view.
   - Applications Panel line in the Source view:
     `fnd:applicationsPanel...
   - Applications Panel entry in the Structure view hierarchy:
     `fnd:applicationsPanel`

When you select the panel as described in this section, the Applications Panel - Property Inspector is displayed.

18.1.3.3 Edit Applications Panel Properties and Components

This section describes how to edit Applications Panel properties and components.

To edit an application panel property:

1. Access the panel as described in Accessing the Applications Panel on a JSF Page.
2. Select the Applications Panel to display the Property Inspector.
3. Follow the instructions in the Property Inspector to modify the panel property.

To edit an application panel component:

1. Access the panel as described in Accessing the Applications Panel on a JSF Page.
2. Select, then double-click the component or subsection. For example, to select a panel header, select `af:panelHeader` in the Source view.
3. When you double-click the component, the Property Inspector for the component is displayed. Edit the component in the Inspector.

   For example, to edit a subsection display name, select the subsection and edit the Text property in the Property Inspector for that subsection.

18.1.3.4 Add a Data Source to an Existing Panel

This section describes how to add a data source after you create a panel in the Application Panel wizard.
To add a data source to an existing panel:

1. In the Application Navigator, open the Data Control panel.

2. Navigate to the data source to bind to the Applications panel.

3. Drag and drop the data source itself (or its individual fields) to the JSF page in Design mode.

Data-source fields are bound to panel components. Components are stored in the contents facet as *af:panelFormLayout* components, and in the various subsections.

For example, Figure 18-13 shows a panel’s Structure view, which contains added components.

![Figure 18-13 Panel Structure View](image)

To create an additional field in a subsection, drag an attribute from the data source to the corresponding container. For example, drag the attribute to *fnd:applicationsPanel > f:facet - contents > af:panelGroupLayout > af:panelFormLayout*. When prompted for the component to associate with the attribute, choose *ADF Input Text w/Label*.

### 18.1.3.5 Add User-Interface Content to Applications Panels

Although Applications panels already provide common layout components, your JSF page might require additional UI elements, such as additional action buttons. When you add new components to a panel, you drop the new components into facets.

To add UI content to existing panels:

1. Open the Components Palette.

2. Drag and drop the button component on to the appropriate popup facet.

   For example, to add a new button, drag and drop the button to the *actionButtonBar* facet.

For more information on facets, see Table 18-2.

### 18.2 Implementing the Simple Panel

The Simple Panel Applications Core declarative component for FUSE Plus is a lighter version of the Applications Panel component. It supports these facets:
• Header Title
• Header Context
• Header Toolbar (Page Actions)
• Horizontal Train
• Content

Built-in buttons for the Simple Panel include:
• Actions (menu button)
• Save and Continue/Save and Close
• Save and Close/Save (Save and Close is the default action while Save is in the dropdown menu)
• Save and Close
• Cancel
• Submit
• Done

This image shows the basic content and appearance of a Simple Panel.

**Figure 18-14  Example Simple Panel**

This table shows the facets supported by the Simple Panel.

**Table 18-3  Facets Supported by Simple Panel**

<table>
<thead>
<tr>
<th>Facet</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contents</td>
<td>Facet for holding the form elements of Simple Panel</td>
</tr>
<tr>
<td>actionButtonBar</td>
<td>Facet for holding the action buttons</td>
</tr>
<tr>
<td>saveButtonMenu</td>
<td>Facet for Save button options</td>
</tr>
<tr>
<td>submitButtonMenu</td>
<td>Facet for Submit button options</td>
</tr>
<tr>
<td>popup</td>
<td>Facet for popup</td>
</tr>
<tr>
<td>panelContext</td>
<td>Facet for displaying context information next to the header</td>
</tr>
<tr>
<td>customSaveDropButton</td>
<td>Facet for adding a custom Save drop button</td>
</tr>
<tr>
<td>panelTrain</td>
<td>Facet for adding a horizontal train above header</td>
</tr>
<tr>
<td>customButtonBar</td>
<td>Facet for holding buttons added using the Application Composer</td>
</tr>
</tbody>
</table>
Table 18-3  (Cont.) Facets Supported by Simple Panel

<table>
<thead>
<tr>
<th>Facet</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>profileImage</td>
<td>Facet for profile image</td>
</tr>
<tr>
<td>secondaryButtonsFacet</td>
<td>Facet for adding customer buttons</td>
</tr>
</tbody>
</table>

This table shows the attributes supported by the Simple Panel.

Table 18-4  Attributes Supported by the Simple Panel

<table>
<thead>
<tr>
<th>Property</th>
<th>Attribute</th>
<th>Type</th>
<th>Defaul t Value</th>
<th>Manda tory?</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>title</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>n/a</td>
</tr>
<tr>
<td>General</td>
<td>panelToolbarRende red</td>
<td>java.lang.Boolean</td>
<td>true</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>General</td>
<td>panelTrainRende red</td>
<td>java.lang.Boolean</td>
<td>false</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>visible property</td>
<td>cancelVisible</td>
<td>java.lang.Boolean</td>
<td>true</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>visible property</td>
<td>saveVisible</td>
<td>java.lang.Boolean</td>
<td>true</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>visible property</td>
<td>saveAndContinueVisible</td>
<td>java.lang.Boolean</td>
<td>true</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>visible property</td>
<td>saveAndCloseVisible</td>
<td>java.lang.Boolean</td>
<td>true</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>visible property</td>
<td>submitVisible</td>
<td>java.lang.Boolean</td>
<td>true</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>visible property</td>
<td>doneVisible</td>
<td>java.lang.Boolean</td>
<td>false</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>shortDesc property</td>
<td>cancelShortDesc</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>n/a</td>
</tr>
<tr>
<td>shortDesc property</td>
<td>saveShortDesc</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>n/a</td>
</tr>
<tr>
<td>shortDesc property</td>
<td>saveAndCloseShortDesc</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>n/a</td>
</tr>
<tr>
<td>Property</td>
<td>Attribute</td>
<td>Type</td>
<td>Default Value</td>
<td>Mandatory?</td>
<td>Valid Values</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------</td>
<td>---------------------</td>
<td>---------------</td>
<td>------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>shortDesc property</td>
<td>saveAndContinue ShortDesc</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>n/a</td>
</tr>
<tr>
<td>shortDesc property</td>
<td>submitShortDesc</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>n/a</td>
</tr>
<tr>
<td>shortDesc property</td>
<td>doneShortDesc</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>n/a</td>
</tr>
<tr>
<td>popupId property</td>
<td>cancelPopupId</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>&lt;id of popup component&gt;</td>
</tr>
<tr>
<td>popupId property</td>
<td>savePopupId</td>
<td>java.lang.String</td>
<td>n/a</td>
<td>No</td>
<td>&lt;id of popup component&gt;</td>
</tr>
<tr>
<td>popupId property</td>
<td>saveAndContinue PopupId</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>&lt;id of popup component&gt;</td>
</tr>
<tr>
<td>popupId property</td>
<td>saveAndClosePopupId</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>&lt;id of popup component&gt;</td>
</tr>
<tr>
<td>popupId property</td>
<td>submitPopupId</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>&lt;id of popup component&gt;</td>
</tr>
<tr>
<td>popupId property</td>
<td>donePopupId</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>&lt;id of popup component&gt;</td>
</tr>
<tr>
<td>rendered property</td>
<td>cancelRendered</td>
<td>java.lang.Boolean</td>
<td>No default value</td>
<td>No</td>
<td>n/a</td>
</tr>
<tr>
<td>rendered property</td>
<td>saveRendered</td>
<td>java.lang.Boolean</td>
<td>n/a</td>
<td>No</td>
<td>n/a</td>
</tr>
</tbody>
</table>
### Table 18-4  (Cont.) Attributes Supported by the Simple Panel

<table>
<thead>
<tr>
<th>Property</th>
<th>Attribute</th>
<th>Type</th>
<th>Default Value</th>
<th>Mandatory?</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>rendered property</td>
<td>submitRendered</td>
<td>java.lang.Boolean</td>
<td>false</td>
<td>No</td>
<td>n/a</td>
</tr>
<tr>
<td>rendered property</td>
<td>saveAndContinueRendered</td>
<td>java.lang.Boolean</td>
<td>false</td>
<td>No</td>
<td>n/a</td>
</tr>
<tr>
<td>rendered property</td>
<td>saveAndCloseRendered</td>
<td>java.lang.Boolean</td>
<td>false</td>
<td>No</td>
<td>n/a</td>
</tr>
<tr>
<td>rendered property</td>
<td>doneRendered</td>
<td>java.lang.Boolean</td>
<td>false</td>
<td>No</td>
<td>n/a</td>
</tr>
<tr>
<td>disabled property</td>
<td>cancelDisabled</td>
<td>java.lang.Boolean</td>
<td>No default value</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>disabled property</td>
<td>submitDisabled</td>
<td>java.lang.Boolean</td>
<td>No default value</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>disabled property</td>
<td>doneDisabled</td>
<td>java.lang.Boolean</td>
<td>No default value</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>partialTriggers property</td>
<td>cancelPartialTriggers</td>
<td>java.lang.Object</td>
<td>No default value</td>
<td>No</td>
<td>&lt;ids of components&gt;</td>
</tr>
<tr>
<td>partialTriggers property</td>
<td>savePartialTriggers</td>
<td>java.lang.Object</td>
<td>No default value</td>
<td>No</td>
<td>&lt;ids of components&gt;</td>
</tr>
<tr>
<td>partialTriggers property</td>
<td>saveAndContinuePartialTriggers</td>
<td>java.lang.Object</td>
<td>No default value</td>
<td>No</td>
<td>&lt;ids of components&gt;</td>
</tr>
<tr>
<td>partialTriggers property</td>
<td>saveAndClosePartialTriggers</td>
<td>java.lang.Object</td>
<td>No default value</td>
<td>No</td>
<td>&lt;ids of components&gt;</td>
</tr>
<tr>
<td>partialTriggers property</td>
<td>submitPartialTriggers</td>
<td>java.lang.Object</td>
<td>No default value</td>
<td>No</td>
<td>&lt;ids of components&gt;</td>
</tr>
<tr>
<td>Property</td>
<td>Attribute</td>
<td>Type</td>
<td>Default Value</td>
<td>Mandatory?</td>
<td>Valid Values</td>
</tr>
<tr>
<td>----------------</td>
<td>----------------------------</td>
<td>------------------</td>
<td>---------------</td>
<td>------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>partialTriggers property</td>
<td>donePartialTriggers</td>
<td>java.lang.Object</td>
<td>No default value</td>
<td>No</td>
<td>&lt;ids of components&gt;</td>
</tr>
<tr>
<td>partialTriggers property</td>
<td>panelPartialTriggers</td>
<td>java.lang.Object</td>
<td>No default value</td>
<td>No</td>
<td>&lt;ids of components&gt;</td>
</tr>
<tr>
<td>partialSubmit property</td>
<td>cancelPartialSubmit</td>
<td>java.lang.Boolean</td>
<td>true</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>partialSubmit property</td>
<td>savePartialSubmit</td>
<td>java.lang.Boolean</td>
<td>true</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>partialSubmit property</td>
<td>saveAndContinuePartialSubmit</td>
<td>java.lang.Boolean</td>
<td>true</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>partialSubmit property</td>
<td>saveAndClosePartialSubmit</td>
<td>java.lang.Boolean</td>
<td>true</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>partialSubmit property</td>
<td>submitPartialSubmit</td>
<td>java.lang.Boolean</td>
<td>true</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>partialSubmit property</td>
<td>donePartialSubmit</td>
<td>java.lang.Boolean</td>
<td>true</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>style property</td>
<td>saveOptionsStyle</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>dropButton/sideBySide</td>
</tr>
<tr>
<td>style property</td>
<td>submitStyle</td>
<td>java.lang.String</td>
<td>dropButton</td>
<td>No</td>
<td>dropButton/&lt;any other string hides the dropdown&gt;</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>submitText</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>n/a</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>submitAccessKey</td>
<td>java.lang.Character</td>
<td>No default value</td>
<td>No</td>
<td>n/a</td>
</tr>
<tr>
<td>Property</td>
<td>Attribute</td>
<td>Type</td>
<td>Default Value</td>
<td>Mandatory?</td>
<td>Valid Values</td>
</tr>
<tr>
<td>------------------------------</td>
<td>--------------</td>
<td>------------------------</td>
<td>---------------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>titleWidth</td>
<td>java.lang.String</td>
<td>500px</td>
<td>No</td>
<td>&lt;valid integer&gt;px or auto</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Use to set the minimum width for the panel title. Generally used to display</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>the title with the minimum width when button overflow is happening. If</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>titleWidth is set to auto, styleClass will be set to &quot;spTitleWidth&quot; on the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>panelHeader component in SimplePanel.jsp. (styleClass &quot;spTitleWidth&quot; is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>part of applcore-altadditions.css and it sets width based on browser</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>dimensions).</td>
</tr>
<tr>
<td>Profile related attribute</td>
<td>profileRendered</td>
<td>java.lang.Boolean</td>
<td>false</td>
<td>No</td>
<td>true/false</td>
</tr>
<tr>
<td>Profile related attribute</td>
<td>profileWidth</td>
<td>java.lang.String</td>
<td>53px</td>
<td>No</td>
<td>&lt;valid integer&gt;px</td>
</tr>
<tr>
<td>Profile related attribute</td>
<td>profileHeight</td>
<td>java.lang.String</td>
<td>65px</td>
<td>No</td>
<td>&lt;valid integer&gt;px</td>
</tr>
<tr>
<td>Title template related attribute</td>
<td>titleTemplate</td>
<td>java.lang.String</td>
<td>single</td>
<td>No</td>
<td>single, objectTypeAndValue, tabbedPage, trainPage</td>
</tr>
<tr>
<td>Title template related attribute</td>
<td>titleObjType</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>String value representing Object Type</td>
</tr>
</tbody>
</table>
### Table 18-4 (Cont.) Attributes Supported by the Simple Panel

<table>
<thead>
<tr>
<th>Property</th>
<th>Attribute</th>
<th>Type</th>
<th>Default Value</th>
<th>Mandatory?</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title template related attribute</td>
<td>titleObjValue</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>String value representing Object Value</td>
</tr>
<tr>
<td>Title template related attribute</td>
<td>titleName</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>String value representing Tab name or Train Stop name</td>
</tr>
<tr>
<td>Title template related attribute</td>
<td>titleObjValueMaxLengthInTrain</td>
<td>java.lang.String</td>
<td>No default value</td>
<td>No</td>
<td>Maximum length of Object Value that can be shown in UI</td>
</tr>
</tbody>
</table>

This table shows the action methods supported by the Simple Panel.

### Table 18-5 Action Methods Supported by the Simple Panel

<table>
<thead>
<tr>
<th>Method</th>
<th>Mandatory?</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancelAction</td>
<td>No</td>
</tr>
<tr>
<td>saveAction</td>
<td>No</td>
</tr>
<tr>
<td>saveAndCloseAction</td>
<td>No</td>
</tr>
<tr>
<td>saveAndContinueAction</td>
<td>No</td>
</tr>
<tr>
<td>submitAction</td>
<td>No</td>
</tr>
<tr>
<td>doneAction</td>
<td>No</td>
</tr>
</tbody>
</table>

This table shows the ActionListener methods supported by the Simple Panel.

### Table 18-6 ActionListener Methods Supported by the Simple Panel

<table>
<thead>
<tr>
<th>ActionListener</th>
<th>Mandatory?</th>
</tr>
</thead>
<tbody>
<tr>
<td>cancelActionListener</td>
<td>No</td>
</tr>
<tr>
<td>saveActionListener</td>
<td>No</td>
</tr>
<tr>
<td>saveAndCloseActionListener</td>
<td>No</td>
</tr>
<tr>
<td>saveAndContinueActionListener</td>
<td>No</td>
</tr>
<tr>
<td>submitActionListener</td>
<td>No</td>
</tr>
<tr>
<td>doneActionListener</td>
<td>No</td>
</tr>
</tbody>
</table>
18.2.1 How to Implement the Simple Panel

Use this sample code as the basis to use the fnd:simplePanel component.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
    xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
    xmlns:fnd="http://xmlns.oracle.com/apps/fnd/applcore"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
    <fnd:simplePanel id="sp1" title="Simple Panel: Title"
        cancelRendered="true" cancelShortDesc="cancel"
        doneRendered="true" doneVisible="true" doneShortDesc="done"
        saveAndContinueRendered="true"
        submitRendered="true" submitText="Submit"
        submitStyle="dropButton">
        <f:facet name="contents">
            <af:panelBorderLayout id="pbl1"
                inlineStyle="border-style:groove; outline-color:Black; height:300.0px;">
                <f:facet name="start"/>
                <f:facet name="bottom"/>
                <f:facet name="end"/>
                <f:facet name="top">
                    <af:outputText value="contents facet here" id="op1" inlineStyle="font-size:large;"/>
                </f:facet>
            </af:panelBorderLayout>
        </f:facet>
        <f:facet name="ActionButtonBar">
            <h:panelGroup id="pg1">
                <af:commandButton text="CustomButton1" id="cb1"/>
                <af:commandButton text="CustomButton2" id="cb2"/>
            </h:panelGroup>
        </f:facet>
        <f:facet name="saveButtonMenu">
            <af:commandMenu id="cm1" text="CustomSaveButtonMenu"/>
        </f:facet>
        <f:facet name="submitButtonMenu">
            <af:commandMenu id="cm2" text="CustomSubmitButtonMenu"/>
        </f:facet>
        <f:facet name="popup"/>
        <f:facet name="panelContext">
            <af:outputText value="(panelContext facet here)" id="ot1"/>
        </f:facet>
        <f:facet name="customSaveDropButton">
            <af:commandToolBarButton text="CustomSaveDropButton" id="cb3"/>
        </f:facet>
        <f:facet name="panelTrain">
            <af:outputText value="panelTrain facet here" id="ot2"/>
        </f:facet>
    </fnd:simplePanel>
</jsp:root>
```

Implement Applications Panels, Master-Detail, Hover, and Dialog Details
18-29
18.2.2 How to Add Simple Panel at Design Time

**Note:**

- As is done with the Applications Panel, the visible attribute and the rendered attribute are set to `true` for each button that is selected at design time.
- For the `submitStyle` attribute, the `dropButton` and `sideBySide` options let the user pick something to indicate the "any other string" that hides the dropdown. `sideBySide` is set as the attribute value to be consistent with the `saveOptionsStyle`.

To add a Simple Panel to your page, follow these steps:

1. As shown here, you can click-and-drag the Simple Panel from the Data Control palette or from the Component palette.

![Figure 18-15 Dragging and Dropping Simple Panel onto Page](image)

When you drop the Simple Panel component, the Simple Panel wizard is started.

2. If you dragged from the Component Panel, you can specify the data source for the component or just click Next to create prototype data.

![Figure 18-16 Select a Data Source](image)
To bind a data source to the panel component:

- Select Bind Data Now.
- Click **Browse** to display the Data Source dialog.

![Data Source Dialog](image)

- Select the data source, then click **OK** to add it to the component. Optionally, you can bind the component to a data source at a later time.

Click **Next** to configure the panel Title.

3. Configure the panel title.

![Configure the Panel Title](image)

**Note:** See Using Simple Panel Title Templates.

There are four title templates from which to choose:

- **Single Title Template**

  A single string can be selected as the Simple Panel title.

  Example Title Format: `<single string>`

  Example: John Jones

  Using the picker drop box, shown here, to the right of the text field, three options are available:
Figure 18-19  Single Title Template Pick Options

- **Select Text Resource.** This will open the same Text panel commonly available in the property inspector. The developer can select an existing translation value, or create a new one. See the Select Text Resource image and description.

- **Text Expression Builder.** This will open a dialog in which the developer can type a value to be translated and allows the insertion of any attribute in the current Collection as bind variables.

- **Clear.** Clear the current field value.

In addition, the user can just type a string into the field. If it exists as an existing Text resource it will be used, otherwise a new resource will be created.

• **Tabbed Page Template**

When Tabbed Page is selected, the options appear similar to this example:

**Figure 18-20  Tabbed Page Title Options**

Three fields can be selected as the Simple Panel title.

Example Title Format: `<tab name>: <object type> <object value>`

**Example:** Suppliers: Negotiation 123456

The options available to the two String fields Tab Name and Object Type are the same as for the Single option, while for Object Value a list of attributes in the current collection is shown. If desired, a text expression can be selected from the pickers.

• **Train Page Template**

When Train Page is selected, the options appear similar to this example:

**Figure 18-21  Train Page Title Options**

Three fields can be selected as the Simple Panel title.

Example Title Format: `<object type> <object value>: <train stop name>`

**Example:** Negotiation 123456: Select Suppliers

The options available to the two String fields Object Type and Train Stop Name are the same as for the Single option, while for Object Value a list of attributes in the current collection is shown. If desired, a text expression can be selected from the pickers. For the Maximum length in train attribute which is
supposed to resolve to an integer number, either enter the number or select from the EL picker which is available. No Text resource picker is provided.

- **Title Object and Value Template**
  When Title Object and Value is selected, the options appear similar to this example:

  **Figure 18-22  Title Object and Value Title Options**

  ![](image)

  Two fields can be selected as the Simple Panel title.

  **Example Title Format:** `<object type> <object value>`

  **Example:** Expense Report: 123456

  The String field Object Type option is the same as for the Single option, while for "Object Value" a list of attributes in the current collection is shown. If desired, a text expression can be selected from the pickers.

  An example of the markup produced is:

  ```xml
  <fnd:simplePanel id="SP1" titleTemplate="objectTypeAndValue"
                   titleObjType="#{viewcontrollerBundle.EMPLOYEE}" titleObjValue="#{bindings.Ename.inputValue}"
                   doneVisible="true" doneRendered="true"
                   panelPartialTriggers="ASPRNnext1 ASPRNprev1 ASPSelId1">
  Click Next.

  4. Select the initial set of panel components.

  **Figure 18-23  Select Panel Components**

  ![](image)

  In the Panel Components dialog:

  Click Read-only Form to create a read-only form. (optional) If you select Read-only, the choices in the Component to Use column will change from Input Text to Output Text types.

  To reorder component fields, click the up and down arrows. To delete component fields, click the Delete icon (X). You will be able to add more component fields later.
• **Display Label:** In general, the labels defined in the selected Data Control will be what you want and you can leave this value at the default <Default> setting. Otherwise, enter a new label name.

• **Value Binding:** In general, the label and the Value Binding will match and you can accept the displayed value. Otherwise, you can click in the field to display a drop-down list of the values available in the selected Data Control.

• **Component To Use:** Data in Dialog Details can be read-only or can be updated. Component to Use is similar to what Component does while creating a table. Clicking it reveals a choice list of values, and the dialog details popup would then at runtime show that particular column from the datacontrol as the selected component to use. The choice list is changed according to whether you choose Read-only. If you selected Read-only, the choices will change from Input Text to Output Text types.

Click Next.

5. Select a navigation button template.

The Navigation Buttons page is designed to allow the selection of different template types.

Each selection allows the specification of different attributes. See the Simple Panel Title templates, above, for definitions.

**Figure 18-24 Select Navigation Button Template**

Four navigation button templates are available:

• **None**
  
  No navigation buttons will be generated. This is the default option unless the title on the title step of the wizard was selected as Train Page. In this case, Train is selected here.

• **Train**
  
  Select this option to provide navigation between page fragments in a bound taskflow using a train.

  If Train is selected, the options resemble this example:
Select Train Navigation Template

To just create the buttons, make sure the Generate train checkbox is not selected.

To bind to a train, check the Generate train checkbox. It is assumed that the page is in a bound taskflow that has train selected and that a train stop display-name has been added for each view activity.

Example markup generated:

```xml
<fnd:simplePanel id="SP1" titleTemplate="trainPage"
   titleObjType="#{viewControllerBundle.EMPLOYEE}"
   titleObjValue="#{bindings.Ename.inputValue}"
   titleObjValueMaxLengthInTrain="40"
   titleName="#{viewControllerBundle.FIRST}"
   saveAndCloseVisible="true" saveAndCloseRendered="true"
   cancelVisible="true" cancelRendered="true"
   panelTrainRendered="true">
   <f:facet name="contents">
     <af:panelGroupLayout id="pgl2">
       <af:panelFormLayout rows="1" id="pfl1">
         ...
       </af:panelFormLayout>
     </af:panelGroupLayout>
   </f:facet>
   <f:facet name="actionButtonBar">
     <af:panelGroupLayout layout="horizontal" id="pgl3">
       <af:commandImageLink id="ASPprev1" icon="/images/applcore/fuseplus/patterns/
          func_arrowheadleft_16_ena.png"
          hoverIcon="/images/applcore/fuseplus/patterns/
          func_arrowheadleft_16_hov.png"
          depressedIcon="/images/applcore/fuseplus/patterns/
          func_arrowheadleft_16_onb.png"
          disabledIcon="/images/applcore/fuseplus/patterns/
          func_arrowheadleft_16_dis.png"
          partialSubmit="true"
          action="#{controllerContext.currentViewPort.taskFlowContext.trainModel.previous == null}"
         />
       <af:spacer width="10" id="s1"/>
       <af:commandImageLink id="ASPnext1" icon="/images/applcore/fuseplus/patterns/
          func_arrowheadright_16_ena.png"
          hoverIcon="/images/applcore/fuseplus/patterns/
          func_arrowheadright_16_hov.png"
          depressedIcon="/images/applcore/fuseplus/patterns/
          func_arrowheadright_16_onb.png"
          disabledIcon="/images/applcore/fuseplus/patterns/
          func_arrowheadright_16_dis.png"
          partialSubmit="true"
         />
     </af:panelGroupLayout>
   </f:facet>
</fnd:simplePanel>
```
• **Linear Record**

Select this option for navigation between data records on the same page.

If Linear Record is selected, the options resemble this example:

**Figure 18-26  Select Linear Record Navigation Template**

A text resource in the style of "1 of 5" is created binding to the current record number and total number of records. This can be changed with the pickers.

Example markup generated:

```xml
<fnd:simplePanel id="SP1" titleTemplate="objectTypeAndValue" titleObjType="#{viewcontrollerBundle.EMPLOYEE}" titleObjValue="#{bindings.Ename.inputValue}" doneVisible="true" doneRendered="true" panelPartialTriggers="ASPRNnext1 ASPRNprev1 ASPSelId1">
  <f:facet name="contents">
    <af:panelGroupLayout id="pgl1">
      <af:panelFormLayout rows="1" id="pfl1">
        ...
      </af:panelFormLayout>
    </af:panelGroupLayout>
  </f:facet>
  <f:facet name="actionButtonBar">
    <af:panelGroupLayout layout="horizontal" id="pgl2">
      <af:commandImageLink id="ASPRNprev1" icon="/images/applcore/fuseplus/patterns/func_arrowheadleft_16_ena.png" hoverIcon="/images/applcore/fuseplus/patterns/func_arrowheadleft_16_hov.png" depressedIcon="/images/applcore/fuseplus/patterns/func_arrowheadleft_16_onb.png" disabledIcon="/images/applcore/fuseplus/patterns/func_arrowheadleft_16_dis.png" partialSubmit="true" actionListener="#{bindings.Previous.execute}" disabled="#{!bindings.Previous.enabled}"/>
    </af:panelGroupLayout>
  </f:facet>
</fnd:simplePanel>
```
• Non-Linear Record
Select this template for data navigation between data records on the same page with the ability to skip across records.

• When this template is selected, the options appear similar to this example:

**Figure 18-27  Select Non-Linear Record navigation Template**

![Non-Linear Record template]

Pick an attribute, either in the current Collection or in a parent Collection (master-detail style) to act as the non-linear record identifier. Click the pick icon by the List Attribute field to display the List Attribute picker:

**Figure 18-28  Selecting the List Attribute**

![List Attribute picker]

Select a collection, such as CreatedBy in the example and click **OK** to place the attribute in the List Attribute field, shown here:

**Figure 18-29  Populated List Attribute**

![Populated List Attribute]

Example markup generated:
Implementing the Simple Panel

Binding generated for a combo box:

6. Select which of the available buttons to display.
Figure 18-30  Select Buttons to Display

- **Editable**
  Select this option if the user needs to be able to edit the content of the panel.

- **Read-only**
  Select this option to prevent changes to the content of the panel. Only a default **Done** button will be made available to the user.

- **No buttons**
  Select this option to display no buttons in the panel.
  
  One use for this option is on work area landing pages. These would have a panel without buttons because they are read-only and not directly transactional. Users can navigate to a new task from the landing page or initiate lightweight transactions that can be performed in a dialog box. Consequently, no final page-level actions appear on the page.

- **Button options**
  The available buttons are grouped into selections. Select the group that contains the buttons and options you want.

  **Submit Button Name**
  
  You can change the name of this button from the default Submit to another name that is more descriptive of the operation.
  
  To change the **Submit** button name, enter the string here. The string will be converted to a text resource and added to the default resource bundle.
  
  If you already have a name defined in a resource bundle, click the edit icon and choose from the list, as shown here.
Figure 18-31  Select Text Resource for Submit Button

- **Resource Bundle**: Select the bundle containing the string you want to use. You can select a single bundle or all available bundles. The strings will display in the Matching Text Resources field.

- **Display Value**: You can enter a new string for a name here.

- **Key**: Each resource must have a unique Key. This generally is, in all upper-case characters, the words of the Display Value separated by an underscore character.

- **Description**: This is an optional entry.

- **Matching Text Resources**: This field displays the entries of the selected resource bundle. Select an existing title from the list.

**Drop down menu**

When this option is selected, the buttons will display in a drop-down menu, instead of side-by-side.

- **Cancel**
  
  This is a default button and cannot be removed.

  Click Next.

7. Select whether or not to enable the side panel.
The side panel is a slide in/out panel on the right in which Oracle Social Network can be located, so you can slide it out and see an Oracle Social Network conversation.

Enable **Generate Additional Draw item** to create another place in the side panel in which to put things. It would get the label provided in the Label field and would be populated in a later design time step.

8. Review the panel structure. When it is correct, click **Finish** to add the Simple Panel to your page.

**Using Simple Panel Title Templates**

The Simple Panel component is enhanced to let developers configure the Page Title so it is appropriate to the product line, page template type, or page context. This section covers the Title templates that are supported, their respective format, and their runtime behavior.

**Supplied Templates**

The supplied templates are:

- **Single string**
  
  A single string has:

  1 field
  
  Title format = <single string>
  
  Example: John Jones
• **Object Type and Value**
   An Object type and Value string has:
   - 2 fields
   - Title format = <object type>:<object value>
   - Example: Expense Report: 123456

• **Tabbed page**
   A Tabbed page string has:
   - 3 fields
   - Title format = <tab name>:<object type> <object value>
   - Example: Suppliers: Negotiation 123456

• **Train page**
   A Train page string has:
   - 3 fields
   - Title format = <object type> <object value>:<train stop name>
   - Example: Negotiation 123456: Select Suppliers

**Note:** The separator character is coded inside the component and cannot be changed.

### Runtime Behavior
For the attributes that support the Title based on a template, see “Title template related attributes” in Attributes Supported by the Simple Panel.

These attributes will be used to support Title based on template:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default Value</th>
<th>Mandatory?</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>java.lang.String</td>
<td>NA</td>
<td>No</td>
<td>NA</td>
</tr>
<tr>
<td>titleTemplate</td>
<td>java.lang.String</td>
<td>single</td>
<td>No</td>
<td>single, objectTypeAndValue, tabbedPage, trainPage</td>
</tr>
<tr>
<td>titleObjType</td>
<td>java.lang.String</td>
<td>NA</td>
<td>No</td>
<td>String value representing Object Type</td>
</tr>
<tr>
<td>titleObjValue</td>
<td>java.lang.String</td>
<td>NA</td>
<td>No</td>
<td>String value representing Object Value</td>
</tr>
<tr>
<td>titleName</td>
<td>java.lang.String</td>
<td>NA</td>
<td>No</td>
<td>String value representing Tab name or Train Stop name</td>
</tr>
</tbody>
</table>
Based on the Title template chosen, the title is shown using these formats at runtime:

### Table 18-8 Title Template Formats

<table>
<thead>
<tr>
<th>titleTemplate Value</th>
<th>Attributes Forming the Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>single</td>
<td>title</td>
</tr>
<tr>
<td>objectTypeAndValue</td>
<td>titleObjType: titleObjValue</td>
</tr>
<tr>
<td>tabbedPage</td>
<td>titleName: titleObjType titleObjValue</td>
</tr>
<tr>
<td>trainPage</td>
<td>titleObjType titleObjValue: titleName</td>
</tr>
<tr>
<td></td>
<td>(For this template, as Object Value comes in the middle and can hold values of bigger length, titleObjValueMaxLengthInTrain can be used by developers to specify the maximum length Object Value can be shown in the UI. If it is specified, Object Value is truncated to the maximum length and shown by appending &quot;...&quot;.)</td>
</tr>
</tbody>
</table>

This table shows examples of attributes and the text that is displayed:

### Table 18-9 Examples of Attributes and Displayed Text

<table>
<thead>
<tr>
<th>Simple Panel Attributes</th>
<th>Title Shown at Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>titleTemplate='single' title='John Jones'</td>
<td>John Jones</td>
</tr>
<tr>
<td>title='John Jones'</td>
<td>John Jones</td>
</tr>
<tr>
<td><em>(titleTemplate not specified)</em></td>
<td></td>
</tr>
<tr>
<td>titleTemplate='objectTypeAndValue' titleObjType='Expense Report' titleObjValue='123456'</td>
<td>Expense Report: 123456</td>
</tr>
<tr>
<td>titleTemplate='tabbedPage' titleName='Suppliers' titleObjType='Negotiation' titleObjValue='123456'</td>
<td>Suppliers: Negotiation 123456</td>
</tr>
<tr>
<td>titleTemplate='trainPage' titleName='Select Suppliers' titleObjType='Negotiation' titleObjValue='123456'</td>
<td>Negotiation 123456: Select Suppliers</td>
</tr>
<tr>
<td>titleTemplate='trainPage' titleName='Select Suppliers' titleObjType='Negotiation' titleObjValue='123456' titleObjValueMaxLengthInTrain=&quot;4&quot;</td>
<td>Negotiation 1234...: Select Suppliers</td>
</tr>
</tbody>
</table>
18.3 Implementing Applications Master-Detail

Note:
Master-Detail refers to the interaction of selecting an object from a master list, and refreshing the details in an adjacent area. It is not the relationship of the data.

The Master-Detail composite is used in situations where the information is too large, dynamic or complex to show in a flat table. The user can see the Master, or summary, information in one area, and the corresponding details in a separate area. This can be achieved using different master and detail components, such as table, tree table, and tree.

For instance, when the user selects an Employee from the master table, the corresponding employee details are displayed in the region below in a label/data format.

For more information, see the Displaying Master-Detail Data section in the Oracle Fusion Middleware Developing Fusion Web Applications with Oracle Application Development Framework guide.

18.3.1 Component Structure and Functions

You should review and understand the Applications Table before proceeding to implement the Applications Master-Detail in your development pages.

Facets

Table 18-10 shows the facets that are exposed on the Applications Master-Detail.

<table>
<thead>
<tr>
<th>Facet</th>
<th>Description</th>
<th>Allowed Children</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>facet for holding the Applications table</td>
<td>formLayout and Applications Table</td>
</tr>
<tr>
<td>Detail</td>
<td>facet for holding the Applications Table</td>
<td>Applications Table</td>
</tr>
</tbody>
</table>

View Properties

See Table 18-2 in Implementing Applications Tables for a list of properties exposed on the Applications Table. These properties can be used to configure the Applications Table under either the Master or Detail section of the Applications Master-Detail component.

Model

The Applications Master-Detail does not expose any bindings to the model on its own, but the ADF tables or formLayout components that are encapsulated within the Applications Table under the master or detail section will be bound to the model.
Controller
The Applications Master-Detail ships a default managed-bean (internal to the Oracle Fusion Middleware Extensions for Applications (Applications Core) team) that currently supports translation functions. You can access the implementation of the Applications Table managed bean which will be exposed as either the Master or the Detail section of the component. For use and implementation information, see Controller in Implementing Applications Tables.

18.3.2 Introduction to Master-Detail Components

The Master-Detail can exist at the page level, or at the subheader level in a page. The Master-Detail component will support these layouts:

- Panel over FormLayout
- Panel over TreeTable
- Panel over Heterogeneous
- Panel over Subtabs
- Panel over Table
- Tree over LevelSpecific
- Table over Table
- Table over Form Layout
- Table over sub tabs
- Table over Heterogeneous (every row can have a different detail section)
- Table over Tree (available via Table over Heterogeneous)
- Form Layout over Table
- Form Layout over Form Layout
- Form Layout over sub tabs
- Form Layout over Heterogeneous
- Form Layout over Tree (available via Form Layout over Heterogeneous)
- Table over Tree Table
- Tree Table over Table
- Tree Table over Sub tabs
- Tree Table over Form Layout

Tables are the most common master component. When a table row is selected, the details appear in the area below the table. A table is also a very common detail component.

A Tree Table is a layout option in a Master-Detail composite for either a Master or a Detail (not both). When a Tree Table row is selected, the details appear in the area below the Tree Table.
Sub tabs are a detail layout option in a Master-Detail composite.
Form Layout is a detail layout option in a Master-Detail composite.

18.3.3 How to Create a Master-Detail

The Applications Master-Detail can be added to a page or page fragment using the Data First approach. Valid drop locations in the page or page fragment include ADF Form, and ADF Layout components and the Applications Panel (jsp:root, af:form, af:root, fnd:applicationsPanel, af:group, af:panelBorderLayout, af:panelBox, af:panelCollection, af:panelFormLayout, af:panelGroupLayout, af:panelHeader, af:showDetailItem, af:panelWindow, af:popup, af:showDetail, af:subform, f:facet, f:panelGrid, f:panelGroup, af:pageTemplateDef, af:pageTemplate#<localArea_Facet>.

For more information on creating a JSF page, see the Oracle Fusion Middleware Developing Web User Interfaces with Oracle ADF Faces guide.

Be sure to save your work after you create each component.

18.3.3.1 Add a Master-Detail to JSF Pages or Page Fragments

You create a Master-Detail in the Applications Master-Detail wizard, which is displayed when you add the details to your JSF pages (or page fragments) from the Data Controls panel.

To add a Master-Detail from the Data Control panel:
For Master-Detail to work, you need to define the model-layer such that the Master and Detail are linked by a ViewLink that establishes a relationship from the Master to the Detail.

1. In the Application Navigator, open the Data Control panel.
2. Navigate to the data source that you want to bind to the Master-Detail.
3. Drag the detail data-source onto the page.
4. In the Create context menu that is displayed, choose Applications > Master-Detail. Figure 18-34 shows the parent-child relationship of the selected Data Control, how the child is dragged to the page, and the Master-Detail option on the Create menu.

Figure 18-34    Example of Master-Detail Relationships
The Applications Master-Detail wizard is displayed.

18.3.3.2 Adding Master-Details Components Using the Applications Master-Details Wizard

This section explains how to use the Applications Master-Detail wizard to add Master-Details to your pages.

All mandatory fields in the wizard contain default values, allowing you to accept the defaults and work through the steps quickly. Clicking Cancel on any of the dialogs cancels the creation of the Master-Detail and does not save the values you entered.

When you click Finish on any of the dialogs, the software:

- Displays a preview of the Master-Detail.
- Creates the Master-Detail with the values you provided on that screen and any previous screens, and default values for the remaining screens. However, not all wizards have a Finish button, or they only appear in a wizard when you have enough information to default the rest of the steps. For example, in almost all wizards, the last step is the Summary. This step often is omitted, so there is almost always a Finish button on the step before the Summary step.

Caution:
Each wizard dialog contains a Messages field that displays errors for that step. Do not proceed to the next wizard step without correcting any errors in the present step.

Creating a Master-Detail Using Tables

When the Create Applications Master-Detail wizard launches, the Select Pattern Type dialog displays, shown in Figure 18-35.

Figure 18-35 Select Pattern Type Dialog
Enclose Master Detail in

If the Pattern Type supports either the Panel Splitter or the Master Header Label, this option becomes active and a list, shown in Figure 18-36, offers these choices.

**Figure 18-36  Enclose Master Detail Choices**

- **<none>**: This is the default selection. No special action will be performed.
- **Panel Splitter**: Select this option to activate the **Splitter Position** field and set the position, in pixels, of the horizontal position of the split. Figure 18-37 shows an example of a Splitter in use at the default position.

**Figure 18-37  Example of Panel Splitter**

- **Panel Header / Show Detail Header**: Select one of these options to activate the **Master Header Label** field to enclose the Master-Detail with a header. There are basically two types of headers:
  - with a hide/show icon
  - without the hide/show icon

In the example in Figure 18-38, the Edit Element Entries text is the Panel Header and the Basic Information text with the expand/collapse icon is the Show Detail Header. The picture, Name and Social Security Number are the content, which is enclosed by the Show Detail Header. Then everything is enclosed by the Panel Header, shown in Figure 18-38.
• **Master Header Label**: Enter the label to be used by either the Panel Header or the Show Detail Header.

Select the Pattern Type (the example uses Table/Table) and any options and click **Next**.

The Configure Master dialog displays, shown in **Figure 18-39**.

**Figure 18-39 Table/Table Configure Master Dialog**

To create your Master-Detail for your Master table columns:

1. **Click Read-only Table** to create a read-only Master table. (optional) If you select Read-only, the choices in the Component to Use column will change from Input Text to Output Text types.

2. In the **Enable ADF Behavior** section, choose whether to allow users to Select, Sort, and Filter rows.
3. In the **Columns** field:
   - Click the **Display Label** field to enter a column label.
   - Click the menu arrow to select value bindings for each value.
   - Choose what component to associate with the column.
   - After selecting a component, click the Delete icon (X) to delete it, the Add icon (+) to edit it, and the Reorder icons to change its position in the field.

Click **Next** to display the Configure Master Table Patterns dialog, shown in Figure 18-40.

**Figure 18-40 Configuring Master Table Patterns**

4. Choose the patterns to be enabled in your table.

5. Click **Next** to display the Configure Detail Header dialog, shown in Figure 18-41.
6. Enter a Detail header name and choose a corresponding attribute.

7. Click Next to display the Configure Details dialog:
   Configuring the Details table is the same as configuring the Master table; see Figure 18-39 and steps 1 through 4.

8. Click Next to display the Configure Detail Table Patterns dialog.
   Configuring the Details Table Patterns is the same as configuring the Master Table Patterns; see Figure 18-40 and step 5.

9. Click Next to display the Review panel configuration dialog, shown in Figure 18-42.

When you click Finish, the Table/Table Master-Detail is added to the editor, and appears similar to Figure 18-43 in Design mode.
Creating a Master-Detail Using Forms

When the Create Applications Master-Detail wizard launches, the Select Pattern Type dialog displays, shown in Figure 18-35.

To create your Master-Detail for your Master form fields or Detail table columns:

1. To create a Master-Detail consisting of two forms, select FormLayout/FormLayout and click Next to display the Configure Master dialog, shown in Figure 18-44.
This configure dialog is the same as the one for creating a Table, except that it does not have the Enable ADF Behavior settings, as shown in Figure 18-39.

2. Click Next to display the Configure Navigation Buttons dialog, shown in Figure 18-45.

![Figure 18-45  Form/Form Configure Navigation Buttons Dialog](image)

Select the navigation buttons you want to appear on the Main form.

3. Click Next to display the Detail Header dialog, shown in Figure 18-41 for details.

4. Click Next to display the Configure Details dialog, shown in Figure 18-46.
Use this dialog to create as many tabs on the Details form as you need. To create a new tab, enter a name in the Name field and click the Add icon. The tab is added in the area beneath the Name field.

Each tab is the same as the dialog for creating a Table, except that it does not have the Enable ADF Behavior settings, shown in Figure 18-39.

5. Click Next to display the Configure Navigation Buttons for the Detail section dialog, shown in Figure 18-45.

6. Click Next to display the Summary dialog.

7. Click Finish to save your changes.

Your new Master-Detail displays in the JSF Page editor, shown in Figure 18-47.
18.3.4 Master-Detail Guidelines for Creating New Records

Developers should follow these guidelines to use the updatable Master-Detail task flow and to investigate some solutions for creating a detail record for a newly-created master record. Several cases have been identified for using Master-Detail. The master and the detail can be a form, table, tree, or tree table.

18.3.4.1 Master-Detail without a Default Primary Key Generator

If you have a Master-Detail in your page and the primary key for the master is not generated using a sequence, the best way to create a detail row for a freshly-created master row is to have a page-level Submit or Save button that must be clicked to save the master data before creating detail data.

18.3.4.2 Master-Detail with a Default Primary Key Generator

In this case there are two solutions:

1. Have a page-level Submit or Save button that would save the newly-created master record before creating a detail record.

2. Set the autoSubmit property on all the elements (components) of the master to true. For example, if the master is a table, set autoSubmit="true" on all the components inside the af:column.

18.3.4.3 Master-Detail with a Composite Primary Key

In this case, you need to provide a page-level Submit or Save button and click it to commit the master record before creating a detail record.

18.3.4.4 Any Other Case

The preferred solution is to have a page-level Submit or Save button that can commit the master record before creating a detail record.
18.3.5 How to Modify Master-Detail Components and Properties

To modify Master-Detail components and properties, double-click a Master-Detail component in the page editor.

18.4 Implementing Hover

Hover is a subset of Detail On Demand that presents the same information when the user hovers over a link.

This is a Design Time (DT) only pattern, no component has been created. For this reason there is no UI First creation option.

The Design Time works when dragging a collection from the Component palette onto an allowed drop component, shown in Figure 18-48.

**Figure 18-48 Dragging from the Component Palette onto a Drop Component**

The allowed drop components are:

- af:commandLink
- af:commandImageLink
- af:commandToolbarButton

The DT will verify whether the component is in a table already bound, and if that binding is for the collection being dropped.

- If so, it will create additional table bindings for the attributes selected.
- If not, it will create form bindings for the collection.

If it is not dragged into an allowed table component, it will create additional bindings for that collection.

If the allowed component already has a showPopupBehavior component child, the menu option will not show. This behavior helps prevent double adding of hovers.

A dialog displays so you can select the attributes to see in the hover popup, and the alignment of the popup over the "hovered" component, shown in Figure 18-49.
The valid values for alignment are:

- afterEnd
- afterStart (default value)
- beforeEnd
- beforeStart
- endAfter
- endBefore
- startAfter
- startBefore

All JSF components created in the popup will be read-only.

When the OK button is clicked:

- The drop component will be given an Id if it does not have one already, and have the clientComponent attribute set to true.
- The drop component will have a `<af:showPopupBehavior>` component added as a child.
- A popup `<af:popup>` will be added as the previous sibling of the drop component, shown in Example 18-2 for the sample markup, and Figure 18-50 for a sample of the result.
Example 18-3 shows the sample markup for a table-based layout and Figure 18-51 shows an example of how the result appears.

**Figure 18-51  Example of a Table-based Layout**

<table>
<thead>
<tr>
<th>MasterId</th>
<th>Col1</th>
<th>Col2</th>
</tr>
</thead>
<tbody>
<tr>
<td>row1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MasterId</td>
<td></td>
<td></td>
</tr>
<tr>
<td>row2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Links in the Popup**

It is possible to add command links / buttons into the popup so the user can navigate to a separate page/page flow. Adding these links is up to the developer, because it is not a valid option in the Design Time, as command links are not an available component in any binder GUI. The developer must ensure the popup is closed after navigation in this case, although the default behavior may do this.

**Example 18-2  Example Markup for a Form-based Layout**

```xml
<af:popup id="popup1">
    <af:panelFormLayout labelAlignment="start">
        <af:panelLabelAndMessage label="# {bindings.MasterId.hints.label}"
            value="# {bindings.MasterId.inputValue}"
            pattern="# {applCorePrefs.numberFormatPattern}"/>
        <af:outputText/>
    </af:panelLabelAndMessage>
    <af:commandLink actionListener="# {bindings.Last.execute}" text="# {applcoreBundle.LAST}" id="rolloverComponent2" clientComponent="true">
        <af:showPopupBehavior triggerType="mouseOver" popupId="popup1" alignId="rolloverComponent2" align="afterStart"/>
    </af:commandLink>
</af:panelFormLayout>
```

**Example 18-3  Example Markup for a Table-based Layout**

```xml
```
18.5 Implementing Applications Dialog Details

The Applications Dialog Details component provides a user interface for launching a popup that contains detail information. The UI can be a detail icon, a link, or a button.

18.5.1 How to Add Applications Dialog Details to Your Page

You can add the Applications Dialog Details to your page in two ways:

- Select the Applications Dialog Details from the Applications component palette and drag and drop it on your page.
- Drag and drop a data control to your page and select the Applications Dialog Details from the list of available UI components.

View

Table 18-11 shows the properties that are exposed on the Applications Dialog Details.
### Table 18-11  Applications Dialog Details Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Allowed Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id (id)</td>
<td>The unique ID for this Applications Table</td>
<td>string</td>
</tr>
<tr>
<td>Rendered (rendered)</td>
<td>Whether the Applications Table is rendered or not</td>
<td>boolean / expression</td>
</tr>
<tr>
<td>Detail Pattern Type</td>
<td>Detail pattern type</td>
<td>image, link or button</td>
</tr>
<tr>
<td>(detailPatternType)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Popup Id (popupId)</td>
<td>ID of the popup to be invoked when Detail image/link/button is clicked</td>
<td>string</td>
</tr>
<tr>
<td>Text (text)</td>
<td>Overrides default label for Detail button, or defines link text for Detail link</td>
<td>expression</td>
</tr>
<tr>
<td>Short Description</td>
<td>Overrides default roll-over text for detail image/button</td>
<td>expression</td>
</tr>
<tr>
<td>(shortDesc)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disabled (disabled)</td>
<td>Sets whether the component needs to be disabled</td>
<td>boolean / expression</td>
</tr>
</tbody>
</table>

### Model

The Applications Dialog Details does not expose any bindings to the model. However, components within the Applications Dialog Details, like the layout inside ADF popup, will be bound to the model.

### Controller

The Applications Dialog Details component does not ship a default managed bean.

#### 18.5.1.1 Adding Applications Dialog Details

The Applications Dialog Details can be added to a page or page fragment using either the Component First or the Data First approach. Valid drop locations in the page or page fragment include ADF Form, and ADF Layout components and the Applications Panel (jsp:root, af:form, af:root, fnd:applicationsPanel, af:column, af:form, af:group, af:panelBox, af:panelFormLayout, af:panelGroupLayout, af:panelHeader, af:showDetailItem, af:panelWindow, af:showDetail, f:facet, f:panelGrid, f:panelGroup, af:pageTemplateDef, af:pageTemplate#<localArea_Facet>).

The Applications Dialog Details can be added to a page or page fragment using either the Component First or the Data First approach. Both approaches launch a wizard which helps you to quickly define the appropriate attribute values. When you complete this wizard, you can further refine the dialog details definition by editing the resulting component as needed.

#### Component First

Navigate to the Component Palette. Click the list of libraries and select Applications. Drag the Applications Dialog Details from the list of components and drop it onto the
page. The wizard will launch after dropping the Applications Dialog Details on the page.

**Data First**

Navigate to the Data Controls panel of the Application Navigator. Open the panel by clicking its bar, then navigate through the hierarchy to locate the data source that you would like to include in the Applications Dialog Details. Select that data source and drag it on to the page. A context menu will appear with a list of components. Move the mouse over the Applications category list. Select **Applications > Dialog Details** to launch the Applications Dialog Details wizard, shown in Figure 18-52.

*Figure 18-52  Launch the Applications Dialog Detail Wizard*

The Applications Dialog Details Create wizard consists of only one panel.

**Create Applications Dialog Details Panel**

The Create Applications Dialog Details Panel will vary depending on the approach used to launch the Applications Dialog Details creation process.

Using the Data First approach, the Bind Data Now and Data Source properties are hidden. The selected data source is automatically bound to the components in the formLayout of the popup.

Using the Component First approach, it is up to the developer to decide whether to bind a Data Source to the dialog details component, shown in Figure 18-53.
You can skip the data control binding step when creating the Applications Dialog Details. In this case, the Applications Dialog Details will create several default placeholder `outputText` fields that you can use for layout purposes in the popup. You can decide how many placeholder columns you wish to display. When you have selected the appropriate number of fields, click **OK** to finish the creation process.

If you wish to bind a data control to the table component using the Component First approach, check the **Bind Data Now** checkbox. This will enable the **Browse** button for the Data Source property. Click the **Browse** button to display a list of data sources available for binding. Navigate through the list, select the desired data source, and click **OK**.

When the Data Source is selected, the developer can enter the title for the popup and choose the Detail Pattern Type.

When **link** is selected for the Detail Pattern Type, you will need to select an attribute of the data source that binds to the Text attribute. This is the displayed text of the link. When **image** or **button** is selected for Detail Pattern Type, choosing an attribute is not needed, as shown in Figure 18-54.

**Figure 18-54 Select a Data Source Attribute**

<table>
<thead>
<tr>
<th>Title</th>
<th>Example Dialog Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detail Pattern Type</td>
<td>link</td>
</tr>
<tr>
<td>Text Attribute</td>
<td>LastName</td>
</tr>
</tbody>
</table>

When the dialog will be read-only:

- If the dialog will be read-only:
The format should be `<Object Type> <Object Name>` (such as Expense Report WBJ3008D)

- If the dialog contains editable fields:
  The format should be `<Action> <Object Type>: <Object Name>` (such as Approve Expense Report: WBJ3008D)

If you want a new Title, enter the string here. The string will be converted to a text resource and added to the default resource bundle.

If you already have a Title defined in a resource bundle, click the ellipsis and choose from the list, as shown in Figure 18-55.

**Figure 18-55 Select Text Resource for Dialog Details Title**

- **Resource Bundle**: Select the bundle containing the string you want to use. You can select a single bundle or all available bundles. The strings will display in the Matching Text Resources field.
- **Display Value**: You can enter a new string for a title here.
- **Key**: Each resource must have a unique Key. This generally is, in all upper-case characters, the words of the Display Value separated by an underscore character.
- **Description**: This is an optional entry.
Matching Text Resources: This field displays the entries of the selected resource bundle. Select an existing title from the list.

When a title is selected from the list, the Title field will appear similar to Figure 18-56.

**Figure 18-56  Dialog Details Title Field Using Resource Bundle**

![Create Dialog Details](image)

<table>
<thead>
<tr>
<th>Title</th>
<th>#{viewcontrollerBundle.WATCHLIST}</th>
</tr>
</thead>
</table>

**Detail Pattern Type**

The Detail Pattern Type is how the data control is shown; it can be an image, a link or a button.

- **Image**: Shows the Dialog Details component as an image. The image is the same as in the component palette. Clicking the image will open the Dialog Details popup.

- **Button**: Shows the Dialog Details component as a button that opens the Dialog Details popup when clicked.

- **Link**: In this case, you will need to select the Text Attribute, which is a list of columns in the Data Control you have selected (or dragged). This column data is used as the link text.

Use of a specific pattern type is your choice and does not affect the way Dialog Details behaves.

- To display what the popup would show, you can choose a link that shows data from the selected data control, such as a column in a table. For instance, in the Employee table, to show more employee data, you can use the employee name as the text attribute. Clicking an employee name then would open more data about that employee.

- A button can be used if there is only one Dialog Details popup for the page. Clicking it gives more information about the data on the page.

- Image could be used the same way as button, or on pages with form data.

**Text Attribute**

This setting is available only if the Detail Pattern Type is link. The text entered here is shown as the Dialog Details link. This helps give the user an idea about the data contained in the popup.

**Read-only Form**

- If content in the dialog is read-only, the window should be non-modal.

- If content in the dialog contains editable fields, the window should be modal.

- If you select Read-only, the choices in the Component to Use column will change from Input Text to Output Text types.

If this option is not selected; that is, the form can be edited by the user, two buttons automatically are added to the form: **Save and Close**, and **Cancel**. Figure 18-57 shows the default buttons in the form in the JDeveloper Design view.
If this option is selected; that is, the form cannot be edited by the user, only an OK button automatically is added to the form, as shown in Figure 18-58.

**Figure 18-58  Default Button Added to Read-only Form**

### Fields

- **Display Label:** This value will be displayed for the column heading. The default value is the text that is displayed in the Value Binding field.

- **Value Binding:** This field lists the names of the columns from the selected data control. Clicking an entry opens a list of the columns so you change the order in which they appear.

- **Component To Use:** Data in Dialog Details can be read-only or updatable. Component to Use is similar to what Component does while creating a table. Clicking it reveals a choice list of values, and the dialog details popup would then at runtime show that particular column from the datacontrol as the selected component to use. The choice list is changed according to whether you choose read-only. If you selected Read-only, the choices will change from Input Text to Output Text types.
18.5.1.2 Working with the Applications Dialog Details

This section discusses how to edit Dialog Details properties.

Editing - Properties

When you have created the Applications Dialog Details, you can modify the property values by using the Property Inspector. There are three ways to select the Applications Dialog Details:

- Select the component in the Design view of the page.
- Select the `<fnd:applicationsDialogDetails ... >` line in the Source view of the page.
- Select `fnd:applicationDialogDetails` from the hierarchy in the Structure View.

All components created as part of the Applications Dialog Details are editable using this same approach, shown in Figure 18-59.

Figure 18-59  Dialog Details Property Inspector

Adding a Data Source

When you have created the Applications Dialog Details, you can see that an `af:popup` has also been created above `fnd:applicationsDialogDetails`. You can add data controls to the facets / content containers within that popup using the following steps:

1. Navigate to the Data Controls panel of the Application Navigator.
2. Open the hierarchy to find the data source.
3. Drag and drop either the entire data source or individual fields into the correct location on the page. The correct location is dependent on the component.

For example, inside the popup that the Applications Dialog Details wizard generates, the fields of the data source are bound to components. Figure 18-60 shows the Structure view of a page with components already added.
To add a field from a data source to the `af:panelFormLayout` inside the popup, drag the field from the data source to the following path: `up > af:dialog > af:panelFormLayout`. As is the case with the data first approach, you will be prompted to choose which ADF component to use for this attribute.

**Note:**
This example uses the Structure view because it provides an efficient overview of the page. The field could also be dropped onto the page in Design or Source view to achieve the same result.

**Adding UI Content**
To achieve the final goals for a page design, you will likely need to add other components to the `af:dialog` inside `af:popup`.

**18.5.1.3 Implementing OK and Cancel Buttons in a Popup**
A product team’s task flow must include the **OK** and **Cancel** buttons that are used to launch a dynamic tab and dismiss the popup, respectively. When the buttons have been added, create a managed bean to set each button’s action listener. Use the method in Example 18-4 as the **Cancel** button’s action listener.
Note:

Although the closePopup() implementation shown in Example 18-4 closes the popup properly, if you reopen the popup by clicking the tasklist link, it shows the previously-entered values. If you do not want to show previously-entered values, you need to add a taskflow return activity, navigate to it, and then close the popup. However, after adding this taskflow return activity to the example closePopup() implementation, the popup is closed only partially. This is a side effect of the Javascript hide that is used. A solution is to use this closePopup() method, which works whether you have the taskflow return activity.

```java
public void closePopup()
{
    FacesContext facesCtx = FacesContext.getCurrentInstance();
    String taskPopupId =
        PatternsUtil.findComponentById(facesCtx.getViewRoot(),
        "TaskPopup").getClientId(facesCtx);
    PatternsPublicUtil.hidePopup(taskPopupId);
}
```

Create another method for the OK button that calls the method in Example 18-4, and any additional processing logic. The common use case would be opening a new task in the Main Area by using the openMainTask API. For example, you can bind the OK button to a managed bean and add your own action listeners, as shown in Example 18-5.

**Example 18-4  Example Method to Create a Managed Bean to Be the Cancel Button’s Action Listener**

```java
import org.apache.myfaces.trinidad.render.ExtendedRenderKitService;
import org.apache.myfaces.trinidad.util.Service;
import javax.faces.component.UIComponent;

public void closePopup(ActionEvent actionEvent)
{
    UIComponent source = (UIComponent) actionEvent.getSource();
    String sourceId =
        source.getClientId(FacesContext.getCurrentInstance());
    ExtendedRenderKitService service =
        Service.getRenderKitService(FacesContext.getCurrentInstance(),
        ExtendedRenderKitService.class);
    String popup =
        "AdfPage.PAGE.findComponent('" + sourceId + 
        ")\'\'.findComponent('::TaskPopup').hide();";
    service.addScript(FacesContext.getCurrentInstance(), popup);
}
```

**Example 18-5  Example Method to Create a Managed Bean to Be the OK Button’s Action Listener**

```java
import javax.el.ExpressionFactory;
import javax.el.MethodExpression;
import javax.faces.event.MethodExpressionActionListener;

public void setOkButton(RichCommandButton okButton)
{
    this.okButton = okButton;
    if(okButton.getActionListeners().length==0)
    {
        FacesContext context = FacesContext.getCurrentInstance();
```
ExpressionFactory ef = context.getApplication().getExpressionFactory();

String methodLink = "#{bindings.openMainTask.execute}";
MethodExpression me =
    ef.createMethodExpression(context.getELContext(), methodLink,
        null, new Class[]{
            ActionEvent.class });
MethodExpressionActionListener methodActionListener =
    new MethodExpressionActionListener(me);
okButton.addActionListener(methodActionListener);

methodLink = "#{pageFlowScope.PopupBean.closePopup}";
me =
    ef.createMethodExpression(context.getELContext(), methodLink,
        null, new Class[]{
            ActionEvent.class });
methodActionListener =
    new MethodExpressionActionListener(me);
okButton.addActionListener(methodActionListener);
This chapter provides guidelines for implementing attachments at design time in a quick and simple manner using Oracle Fusion Middleware components. This chapter includes the following sections:

- Introduction to Attachment Components
- About Creating Attachments
- About Displaying Attachments for Multiple Entities in the Same Table
- About Configuring Attachments Component Properties
- About Working with Attachments Programmatically
- About Setting Up Miscellaneous Attachments Features
- About Integrating Attachments Task Flows into
- About Securing Attachments
- Understanding How to Use Attachments (Runtime)

For general information about Oracle Fusion Middleware user interface (UI) components, see:

- Working with Page Definition Files for an Integrated Excel Workbook Oracle ADF chapter in the Developing Fusion Web Applications with Oracle Application Development Framework
- Introduction to ADF Faces Rich Client chapter in the Developing Web User Interfaces with Oracle ADF Faces

19.1 Introduction to Attachment Components

Attachment components provide a declarative programming mechanism for you to add attachments to the user interface (UI) pages that you create for Fusion web applications. After it is added to a UI page, the component gives users the ability to associate a URL, desktop file, repository file or folder, or text with a business object, such as an expense report, contract, or purchase order.

There are four Attachment components available for use in your page. Attachment Views, Attachment Link, Attachment, and Attachments Carousel. The Attachment Views and Attachment Link components incorporate the latest design concepts. The Attachment component was the original component available.
**Note:**

The Attachment and Attachments Carousel components are still supported. Any UI enhancements being delivered will not be applied to these components.

**Note:**

All Attachments of type File, Folder or Text are stored on a WebCenter Content Server. You will be asked to establish a connection to that server in the pre-requisite configuration steps.

### 19.1.1 About the Attachment Views Component

The attachment views component is a declarative component designed specifically for use with Fuse+. It can be shown in a page or page segment.

The following figure shows what the component looks like, before the advanced features are enabled.

**Note:**

When first accessed, the component displays in table format. Clicking the icons in the top right-hand corner toggles among the table, list and card views.

#### Figure 19-1 Attachment Views Component

![Attachment Views Component](image)

The following element is rendered:

- **Toolbar**
  - Add button - opens the Add Attachments pane
  - Delete button - deletes the selected rows

**Note:**

There is no concept of a selected card in the Cards view. Therefore, Delete Attachment will not appear in the Actions menu for that view.

- **Views**

  There are three views available:
– Table view - shows the Attachments in a table format. Currently showing columns for Type (icon), Title, and Description. Title will be shown as a link when there is a preview version of the Attachment to view. shows the table view.

– List view - shows the Attachments in a list format. Currently showing Attachment values of Thumbnail, Type (icon), Title, Description, Actions. You can click Type to download the Attachment. Title will be shown as a link when there is a preview version of the Attachment to view.

* Cards View — Shows the Attachments in a card format. Currently showing Attachment values of Thumbnail, Title, and Actions. Title will be shown as a link when there is a preview version of the Attachment to view.

• Add Attachments pane - allows the user to provide the minimum details required to create an Attachment (Attachment Type, Category and File, Text or URL)
The Add Attachments pane stays open to allow the user to continue to add attachments. When finished, the user can hide the pane by clicking the close icon (X), located in the top right-hand corner of the pane.

Valid Attachment Types are File (default), Text, URL, and Repository File/Folder.

- File: The user can select multiple files can be selected either through file browser, or by dragging and dropping onto the drop area. The selected files will automatically be uploaded and added. The contents of a compressed file can now also be added automatically by checking the Unzip all compressed files after upload checkbox. By default the checkbox is unchecked allowing the compressed files to be uploaded, but not extracted.

- Text or URL: The user enters the text or URL and then clicks **Add** to create the Attachment.

- Repository File/Folder: The same document picker used with the Attachments component is shown in the Add Attachments pane. The user navigates through the virtual folder structure to find files or folders. When a file is found the user selects it and then clicks **Select** at the bottom of the table to add it as an Attachment. There is also the option to search for files using a query. It is initiated by clicking on the Advanced link at the top of the table. This opens the Advanced Search dialog, as shown in the following figure.
Use the keywords field to perform a search on the contents of the files in Oracle WebCenter Content. In order for this search to be successful, the Content Server must be configured for a full content search. See "Managing Search Tools" in the Oracle WebCenter Content System Administrator’s Guide for Content Server for options.

Additional query filters are enabled when a content type other than "All Content Types" is selected. The other options in the list are all Oracle WebCenter Content profiles. These profiles allow an administrator to control the fields that appear, and the order in which they appear. Selecting this option determines the fields that you can add as additional query filters. A targeted profile limits the number of fields from which to choose to only those that are relevant. A global profile showing all fields is also available.

- Details pane - provides additional details about the current Attachment. You can edit any updateable value of the Attachment in this pane.
Open the pane by clicking the right-arrow button associated with the Attachment. Close the pane by clicking the right-arrow button again, or by clicking the "X" in the top-right corner.

The user controls the changes made. Clicking **Update** saves the edits. Changes are discarded when the pane is closed (by clicking **X**), or when the current Attachment is changed.

File Attachment types are updateable. If a file is already checked out, it will be checked out again after it has been updated. To ensure against file loss in the event of a high-availability failover, a file updates immediately. Discarding changes does not apply to file updates.

The "More Information" section displays extra attributes from the Content Server that are not already in use by Attachments.

When files are stored on the Content Server, it may be necessary to include different formats of the original document for distribution purposes. For example, an Adobe PDF version of a Word file or images that are uploaded in various sizes. Content Server supports this through renditions. The "Rendition" section displays all of the different renditions that have been added to the original file on the Content Server.

Edit text button (text Attachments only) - downloads and displays the text for the Attachment. The text is editable for the remainder of the session. A warning displays when text files are larger than 1 MB.

Editable text displays when you click **Edit**, as shown in the following figure.
19.1.2 About the Attachment Link Component

The Attachment Link component renders these elements on the page or page segment:
The Attachment Link component provides a link to the most recently attached document or URL. Where there is a preview version of the attachment, this will show when clicking this link. Hovering your mouse on the link supplies a detail window with attachment information. The detail window contains the following information about the most recent attachment:

- Type – represented by an icon for File/URL/Text/Folder
- link to the most recently attached document or URL
- Preview, where a preview view is available.
- Last Updated By
- Last Updated Date

If there are no attachments, the value of this field will be None.

The Add Attachment icon. When clicked, the Attachments window is launched showing any existing attachments. The Add pane will be displayed allowing the user to start adding attachments.
When there are one or more attachments, a folder icon will be displayed. Clicking this link opens the Attachments window, which displays the attachments.

Hovering over the folder icon will display a small dialog with a list of up to the next three most recent attachments. When clicked, the link (by default) will prompt the user to download the file. If there is only one attachment, there will be an entry indicating No More. If there are more than four attachments, the last bullet is a link with the format "<# of remaining attachments> more...". The number of attachments shown in the small dialog list is be configured. Clicking this link opens the Attachments window, which displays the full list of attachments in a table.

The Delete Attachments icon. This is shown only when there is an attachment. It allows the most recent attachment to be deleted without having to launch the Attachment window.

**Note:** The Attachment label originates from outside the Attachment component to allow the Attachment component to correctly align with the other components in the page layout.

The Attachment Window displays the Attachments Views component. The popup includes an **OK** button, which closes the popup, and a **Cancel** button. On opening the popup, a savepoint in the transaction will be created. Clicking **Cancel** will initiate a rollback to that point and revert any changes made to Attachments on the database.

**Abridged Mode**

The Attachment Link component also can display with a smaller footprint on the page. In Abridged mode, the component will show a single icon dependent on the number of attachments.

- No attachments: The Add Attachment icon will display. Clicking the icon will open the Attachment window with the Add panel displayed.
- One attachment: A document icon will display.
- More than one attachment: A folder icon will display. Clicking the file or folder icon will display the Attachment window.

**19.1.3 About the Attachment Component**

The component can be displayed in a UI page in any of the following ways:

- **Attachment field in a page or page segment:**
  
  The following elements are rendered on the page or page segment:

  - A link to the most recently attached document or URL. Hovering your mouse on a link supplies a detail window with attachment information. The detail window contains the following information about the most recent attachment:

    * Type - File/URL/Text
    * Attached By
    * Attached Date

    When clicked, the link (by default) will prompt the user to download the file.

    If there are no attachments the value of this field will be **None**.
– When there are more attachments, a link will be displayed in the format "<# of additional attachments> more...". Clicking this link opens the Attachments window, which displays the full list of attachments in a table. Hovering over this link will display a small dialog with a list of up to the next three most recent attachments. When clicked, the link (by default) will prompt the user to download the file. If there are more than four attachments the last bullet is a link with the format "<# of remaining attachments> more...". The number of attachments shown in the small dialog list is configurable. Clicking this link opens the Attachments window, which displays the full list of attachments in a table.

– The **Manage Attachments** icon. When clicked, the Attachments window is launched showing any existing attachments as well as a new empty row at the top of the list to allow the user to add new attachments.

– The **Delete Attachments** icon. Only shown when there is only one attachment added. Allows the one attachment to be deleted without having to launch the Attachment window.

*Figure 19-9* shows an example of an attachment field in a page or page segment with attachments.

*Figure 19-9  Attachment Field in a Page or Page Segment*

Note:
The Attachment label originates from outside the Attachment component to allow the Attachment component to correctly align with the other components in the page layout.

• **Attachment column in a table:**
The following elements are rendered in the table:

– A column header titled Attachments

– A link to the most recently attached document or URL. Hovering your mouse on a link supplies a detail window with attachment information. The detail window contains the following information about the most recent attachment:

  * Type - File/URL/Text
* Attached By
* Attached Date

When clicked, the link (by default) will prompt the user to download the file. If there are no attachments the value of this field will be None.

- When there are more attachments, a link will be displayed in the format "<# of additional attachments> more...". Clicking this link opens the Attachments window, which displays the full list of attachments in a table.

Hovering over this link will display a small dialog with a list of up to the next three most recent attachments. When clicked, the link (by default) will prompt the user to download the file. If there are more than four attachments the last bullet is a link with the format "<# of remaining attachments> more...". The number of attachments shown in the small dialog list is configurable. Clicking this link opens the Attachments window, which displays the full list of attachments in a table.

- The Manage Attachments icon. When clicked, the Attachments window is launched showing any existing attachments as well as a new empty row at the top of the list to allow the user to add new attachments.

- The Delete Attachments icon. Only shown when there is only one attachment added. Allows the one attachment to be deleted without having to launch the Attachment window.

Figure 19-10 shows an example of an attachment column in a table.

Figure 19-10  Attachment Column in a Table

<table>
<thead>
<tr>
<th>Invoice Header 12345</th>
<th>Save and Close</th>
<th>Cancel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purchase Order: 12345</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operating Unit: John</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Terms Date: 21-Sep-2006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invoice Amount: 1,656.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invoice Number: 123</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invoice Date: 04-Apr-2006</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Attachment table:

The Attachment table can be shown in:

- A page or page segment

This occurs when you choose to display all attachments for the current record in a table.

- A dialog

Occurs when:
* The user clicks the **Manage Attachments** icon associated with an attachment field either on a page or page segment, or an attachment column in a table

* The user clicks **Add** or **Edit** in the toolbar of the Attachments Carousel component

The following elements are rendered in the Attachment table:

- **Table Toolbar**
  - **Add** button, which adds a new row at the top of the table
  - **Delete** button, which deletes the selected rows from the table
  - **Set Primary** button, which sets the selected row to be the primary attachment for its category. Displays when the setting the `enablePrimaryCategory` property attribute for the Attachment component to `true` (this button is optional)

- **Reorder Attachments** (**Actions** Menu)
  Launches a window that allows you to modify the order of the Attachments. It displays when the `enableOrdering` attribute has been set to `true`. The objects being ordered are defaulted from the label attribute.

  **Figure 19-11** shows an example of the window.

  **Figure 19-11  Reorder Attachments**

- **Set Primary** (**Actions** Menu)
  Sets the selected row to be the primary attachment for its category. It displays when you set the `enablePrimaryCategory` property attribute for the Attachment component to `true`.

- **Type** (required field that determines the type of Attachment)
  - **File** (default)
Category values are defined at implementation time. Make sure to use functionally relevant category names. If two or more categories are defined, the category column is displayed. If only one category is defined, the column is not displayed.

One attachment out of all the attachments that have the same category value can be designated as the primary. This is option can be enabled by setting the `enablePrimaryCategory` property to true. By default it is off. When enabled a new column title Primary will appear before the Category column. The primary attachment will be indicated by a tick mark.

File Name or URL (required field)

If the value is an existing attachment, a link is shown. When clicked, the link (by default) will prompt the user to download the file. For new rows:

* If Type is File, will show a file upload field
* If Type is Text, will show a text field
* If Type is URL, will show a text field
* If Type is Repository File/Folder, will show a text field and browse button. Clicking on Browse will launch a document picker for finding files in the repository.

Title

The user name for the attachment, as the file name or URL may not adequately convey the contents of the attachment. If users do not enter a title, it defaults to the value in the File Name or URL field.

Description

The field to include additional information on the attachment.

Attached By

Shows the name of the user that last updated the attachment relationship.

Attached Date

Shows the date on which the attachment relationship was updated.

Figure 19-12 is an example of an Attachments table with the `repositoryMode` attribute set to `false`. 
19.1.4 About the Attachments Carousel

The attachments carousel is a popular component for browsing through images. It integrates the attachments and carousel components, simplifying the uptake work for pages that have a heavy image usage with attachments. It can be shown in a page or page segment.

The following elements are rendered in the Attachment table:

- **Toolbar**
  - Add button - opens the Attachment Table in a dialog with a new row.
  - Edit button - opens the Attachment Table in a dialog
  - Delete button - deletes the current attachment item shown in the carousel

The toolbar can be hidden by setting the carouselToolbarRendered property to false.

- **Carousel and CarouselItem**

The carousel controls the display of the attachments. The CarouselItem stamps out the same display settings for each attachment item. All of the properties for the standard Carousel and CarouselItem components are available for you to control the display and behavior of the Carousel.
Any attachment that cannot be displayed as an image will show as its title. The current item in the carousel is also linked. When clicked, the link (by default) will prompt the user to download the file.

**Note:**

The current implementation is showing the actual file and not a thumbnail or preview version. Until thumbnail support can be introduced, it is recommended that users consider the size of the files that are adding for display inside the carousel.

- **Detail**
  The detail section contains the following information about the most current attachment:
  - Type - File/URL/Text
  - Attached By
  - Attached Date

The detail section is hidden by default. However, it can be displayed by setting the attachmentsDetailsRendered property to `true`.

### 19.2 About Creating Attachments

This section provides information about creating attachments and describes how to set up your model project for attachments, how to create an attachment field or an attachments table, and an attachments column in an Applications table. Also included is how to set up required properties, as well as information about what happens when you implement attachments and what happens at runtime.

#### 19.2.1 Performing Prerequisite Tasks

Before you start creating attachments, do the following:

1. Add the *Applications Core (Attachments Model)* library to your Model project:

2. After adding the library to your model, navigate to Application Navigator > Application Resources > Descriptors > META-INF and add the following XML snippet to your `jazn-data.xml` file:

   ```xml
   
   Note:
   Be aware that the URLs and `<class>` entries cannot contain any spaces or line breaks.
   
3. Create the view object representation of the business object that requires attachments in your model project.

4. Create a Content Repository connection in JDeveloper.
Note:

• Attachments stores the name of the Content Repository connection used when creating a new attachment. This value is then used when retrieving the attachments. It is important to co-ordinate the naming and registration of Content Repository connections. So that the Attachments code can connect to the correct server to retrieve the requested file.

• Developers need to create a Content Repository connection with the name "FusionAppsContentRepository". This connection should used jaxws as the socket type and Identity Propagation for Authentication.

For information about how to create a Content Repository connection in JDeveloper, see Creating a Content Repository Connection.

For additional information see the Oracle Fusion Middleware Developing WebCenter Portal Assets and Custom Components with Oracle JDeveloper guide.

5. Do the following to add libraries and configurations that are required to support working with the Content Repository connection:

a. Select a JSF page in the View Controller project that is being enabled for Attachments. Changes made to this page must be manually removed at the completion of this pre-installation step.

b. Navigate to Application Navigator > Application Resources > Content Repository and select the Content Repository connection defined in Step 4. Then, drag it onto the page you selected in Step a.

c. On the context menu that is displayed, choose Document Library and remove the changes made to the page.

Note:

• An alternative to modify an existing page is to create a new JSF page in the project, and then delete the page when you get to Step 5.

• At the time this process was documented, these steps changed the following files in a typical Application. If these files were not been changed, it might have to check them out manually and repeat Steps a through c.
  - Application Resources/Descriptors/ADF META-INF/adf-config.xml
  - Application Resources/Descriptors/META-INF/orion-application.xml
  - Projects/View Controller (that is, the name of your ViewController.jpr file)
  - Projects/View Controller/Web Content/WEB-INF/orion-web.xml
  - Projects/View Controller/Web Content/WEB-INF/web.xml
6. Log onto the Content Repository and confirm that the following has been created:

- *Attachments* folder under the *Contribution Folders* folder
- *Attachments* security group used for unshared attachments
- *SharedAttachments* security group used for shared attachments
- *AttachmentsUser* role with no permissions set
- *AttachmentsRead* user role with read (R) permission to the Attachments security group for unshared attachments
- *AttachmentsWrite* user role with write (W) permission to the Attachments security group for unshared attachments
- *AttachmentsDelete* user role with delete (D) permission to the Attachments security group for unshared attachments
- *AttachmentsAdmin* user role with read, write, delete, and admin (RWDA) permissions to the Attachments security group for unshared attachments
- *SharedAttachmentsWrite* user role with write (W) permission to the SharedAttachments security group - for shared attachments
- *SharedAttachmentsDelete* user role with delete (D) permission to the SharedAttachments security group - for shared attachments
- *SharedAttachmentsAdmin* user role with read, write, delete, and admin (RWDA) permissions to the SharedAttachments security group - for shared attachments.

**Note:**
This setup is required for Attachments to function correctly. You must log in to the Content Server as a user with System Administrator privileges to see the above information. If you cannot see these details in your Content Server, please contact your system administrator to ensure that they have enabled the FusionAppsAttachments component in the Content Server.

7. Set the maximum size for files that can be uploaded. This setting is made to your Application and will apply to any page that uses a multi-part form to upload from the desktop to the Oracle WebLogic Server, not just Attachments.

This step does not need to be completed before continuing with the rest of the implementation. By default the maximum size is defaulted to 2MB, which allow an end-to-end implementation to be completed. However, it is important to modify this setting before deployment. When deciding on the maximum size, considering the type of files that your typical customer will be uploading with your Application will help you determine a value.

For information about how to apply the settings, see "Setting Parameters to Upload Files to Content Repositories" in *Developing WebCenter Portal Assets and Custom Components with Oracle JDeveloper*. 
Related Links
The following document provides additional information related to subjects discussed in this section:

- For more information, see the chapter on managing security groups, roles, and permissions in *Administering Oracle WebCenter Content*.

Example 19-1  Snippet for jazn-data.xml File

```xml
<grant>
  <grantee>
    <principals/>
    <codessource>
      <url>file:${atgpf.oracle.home}/atgpf/modules/oracle.applicore.attachments_11.1.1/Attachments-Model.jar</url>
    </codessource>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
      <name>context=SYSTEM,mapName=oracle.wsm.security,keyName=*</name>
      <actions>read</actions>
    </permission>
    <permission>
      <class>oracle.security.jps.service.keystore.KeyStoreAccessPermission</class>
      <name>stripeName=owsm,keystoreName=keystore,alias=*</name>
      <actions>read</actions>
    </permission>
  </permissions>
</grant>

<grant>
  <grantee>
    <principals/>
    <codessource>
      <url>file:${atgpf.oracle.home}/atgpf/modules/oracle.applicore.attachments_11.1.1/ContentServer-Model.jar</url>
    </codessource>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
      <name>context=SYSTEM,mapName=oracle.wsm.security,keyName=*</name>
      <actions>read</actions>
    </permission>
    <permission>
      <class>oracle.security.jps.service.keystore.KeyStoreAccessPermission</class>
      <name>stripeName=owsm,keystoreName=keystore,alias=*</name>
      <actions>read</actions>
    </permission>
  </permissions>
</grant>
```

19.2.2 Understanding Content Repository Prerequisites

The attachment views component can display more detail from the Oracle WebCenter Content repository. Enabling this functionality, however, requires additional
configuration of the repository. Since the attachment views component is not dependent on this configuration, you can do it when required.

**Thumbnails**

Thumbnails are small preview images. The attachment views component shows thumbnails in the list view, and for any renditions generated in the Details pane. For information about generating thumbnails, see setup steps for "Setting Up Thumbnails" in *Managing Oracle WebCenter Content*.

By default, thumbnails are displayed as 100 x 100 pixels. For information about how to change the default size, see "Changing the Size of Thumbnails" in *Managing Oracle WebCenter Content*.

**Digital Asset Management**

Digital asset management defines and provides specified formats and sizes from the original file for images, video and audio. A defined *rendition set* determines all of the conversions that must be performed. When an Attachment is added, or updated, the default rendition set is applied.

Rendition sets can be defined to create a thumbnail rendition. Consider this before enabling thumbnails and renditions for file formats that will also be processed by rendition sets.

For more information about digital asset management, see "Working With Image and Video Conversions" in *Managing Oracle WebCenter Content*.

For information about how to change the default image rendition set (DefaultPackedConversionSet) see "Modifying the Content Repository Configuration File" in *Managing Oracle WebCenter Content*.

### 19.2.3 About Setting Up Your Model Project for Attachments

The Model project must be set up correctly before you can add attachments to your page.

Part of the Model setup involves creating an attachment view link between the business object view object and the attachments view object. When defining this attachment view link, you must select *Primary Key* values and enter an *Attachment Entity Name*. This unique name is used to identify all attachments for your business object.

The first time you create an attachment view link for your business object, the unique attachment entity name is stored in the database. If you need to link the same set of attachments to another view object on a different page, reselect the existing attachment entity name when creating the attachment view link.

After creating the attachment view link, you will need to assign a set of categories to your attachment entity. When creating the attachment view link, you can choose to show all of the categories assigned to your entity in your UI, or a subset of these categories. If you do not need to reduce the list of categories that will be displayed in your UI, you can select to "Show All Categories" on the Categories step in the Attachment View Link Wizard. This will make all categories that are assigned to your attachment entity available in your UI. For more information, see *How to Create Attachment View Links*.

After you have finished setting up your Model project, you must decide how you would like to display the attachments in your page: As an attachment field, as a table of attachments, an attachment column in an applications table, or as an attachment carousel. This design-time setup is performed in your ViewController project. For
more information about setting up your ViewController project for attachments, see How to Create an Attachments Field_ Attachments Table_ Attachments Carousel_ or Attachment Views, How to Create an Attachments Column in an Applications Table, and About Displaying Attachments for Multiple Entities in the Same Table.

After adding attachments to your UI page, you must complete the required steps to ensure that the attachments functionality works correctly at runtime. For more information about these setup steps, see What Happens When You Implement Attachments and What Happens at Runtime.

19.2.4 How to Create Attachment View Links

Attachment view links are used to establish master-detail relationships between your view objects and the attachments view object. Attachment view links are created using the Attachment View Link wizard.

To create an attachment view link, do the following

1. In JDeveloper, choose Application Navigator > Model Project. Right-click and choose New from the menu to open the New Gallery.

2. Choose the Business Tier > ADF Business Components category. Select the Attachment View Link item, as shown in Figure 19-14.

   Figure 19-14    New Gallery Dialog

3. Click OK to access Step 1 of the Create Attachment View Link wizard, as shown here.
4. Select the New View Link option to Create a New View Link. Click Next. The Name dialog appears as shown here.
5. Complete the following information:

a. Select the name of the module that your view object belongs to.

   Seed data can be created as part of the Wizard flow. Any seed data created will be striped by the provided module so that it can be extracted and shipped as part of deploying your Application. Any existing seed data belonging to your module will also be displayed in the wizard steps.

b. Select the package where the attachment view link will reside.

   The new View Link will be created in this package.

c. Enter a unique name for the attachment view link.

6. Click **Next** to access Step 3. The View Object dialog appears, as shown here.
7. Enter the name for the view link accessor that will be added to the data control of selected view object. Then, from the **Available** column, select the view object that you want to create attachments for.

**Tip:**

There is no need for you to select a destination view object because the Attachment view link always uses the Attachments view object as the destination view object.

8. Click **Next** to access Step 4. The Attachment Entity dialog appears, as shown here. The Attachment Entity is used to uniquely identify your business object.
9. Select an existing Entity Name or create a new Entity Name.

**Note:** The Entity Name is used to map an entity object to its attachments. An entity object can be used across many business objects. In the figures the business object is for a Department. If there were a Department to Employee business object. A View Link could also be created for that business object as well. As the Department Number is also in that table as a foreign key. That View Link would also use the same Entity Name.

**Note:** Historically the name of the primary database table the business object is based on is used.
Where the Entity Name has already been created, click **Browse...** to open the List of Values to search and select. The Selected column will be automatically updated with the stored primary key values from the selection.

For a new Entity Name, click **New...**. You will be prompted to supply an Entity Name, User Entity Name and Data Object Code. The Data Object Code ideally should be the database table. Use the data object code, primary keys and the data collected to create an Attachment. A SQL query could be generated to return the row. Click **OK** to close the window.

**Available / Selected:** Select only those columns that make up the primary key of the entity object by shuttling them from the Available column to the Selected column. Making changes to the Selected columns for an existing Entity Name will update the stored entries.

10. Click **Next** to access Step 5. The Categories Mapping dialog appears, as shown in here. An Entity Name must have at least one Category assigned. This dialog will facilitate mapping Categories to the Entity Name from Step 4.
Click **New**... to create a new Category. A popup will appear to provide the Category Name and Category User Name. This new Category will be stored in the database and stripped by the Module provided in Step 2.

Click **Add**... to choose from other Category values that have already been created for your Module. This list will also include a Miscellaneous category that is centrally-seeded by Attachments. If you cannot anticipate the Category values a customer may want to use with Attachments, choose the Miscellaneous category.

Click **Edit**... to modify the Category User Name for a Category.

Click **Remove**... to remove a Category from being mapped to the Entity Name.
Note: It is not necessary to re-enter the wizard to make changes to the categories or mapping. Management pages are available for administering this information during development and production. For more information, see About Assigning Categories to the Attachment Entity.

11. Click Next to access Step 6. The Categories dialog appears, as shown here. (The Display All Available Categories checkbox is not selected when you first enter this page.) The View Link created can be restricted to a subset of the categories mapped to the entity name in Step 5. This is useful where your functionality needs to ensure that only Attachments of a specific category will be shown and managed using this View Link.

Figure 19-20  Create Attachment View Link Wizard — Categories (Step 6)

Select the Display All Available Categories checkbox to place no restrictions on the categories the View Link can use.
To control which categories are available to the View Link, deselect the Display All Available Categories checkbox. Then move the mapped categories required in the Available column to the Selected column.

The selected list of categories will be stored as a custom property on the newly created attachment view link. The value will consist of a concatenated string of values. All the selected categories are concatenated in a comma-separated list.

**Note:** The custom property on the view link cannot be customized. A customer can map custom categories to your entity name in production. But the new category won’t be seen in any UI that uses a view link with the custom property set.

**Note:** If the all of the categories in the custom property are removed from the mapping, the view link will no longer return any results.

You should choose the Display All Available Categories option to allow for customization of the Category list without code changes. Additionally, be aware that the list of available categories, and the list of Attachments displayed, can be further controlled using Category data security. For more information, see About Attachment Category Data Security

12. Click Next to open the Application Module dialog, as shown here. For the attachments view object to be available to include in your pages, it needs to be registered in the Application Module Data Model. Select Add or Modify an Application Module to create a new instance of the view object that you selected on the View Object page, and create an instance of the attachments view object as a child of the new instance.
Click **Browse**… to open a popup to help select the Application Module to update.

Click **New**… to create a new entry in the table. The attachments view object also includes its own child collections for Extra Attributes and Renditions. These collections are required for showing additional information in the Attachment Views and Attachment Link components. Select Enable Extra Attributes or Enable Renditions based on whether it is required to show this information. If additional custom metadata is to be stored with the attachment then select the Enable Flexfields option.

Click **Remove** to remove the selected View Link Usage from the Application Module.

13. Click **Next** to access the Summary page, shown here. Review your selections and click **Finish** to create your attachment view link.
14. If you chose to keep the Application Module option unselected in Step 10, you need to add your newly-created attachments view link to your application module data model as follows:

a. Choose **Application Navigator > AM name**. Right-click and choose to open the application module in the editor.

b. Choose Data Model from the list of categories to open the Data Model Components dialog.

c. Select the newly created Attachment View Link located in the left column and select the view object that you created the view link for in the right column. Shuttle the Attachment View Link over to the right column.

d. To enable Extra Attributes, select the Attachments Extra Attribute view object located in the Available View Objects (left column). Select
AttachmentsExtraAttributeVO, by opening oracle.apps.model.Model > oracle.apps.fnd.appcore.attachments.Model > oracle.apps.fnd.appcore.attachments.uiModel.view > AttachmentsVO. In the Data Model (right column), select the Attachments View. Shuttle the AttachmentsExtraAttribute View Link over to the right column.

e. To enable Renditions, select the Attachments Attachments Rendition view object located in the Available View Objects (left column). Select AttachmentsRenditionVO, by opening oracle.apps.model.Model > oracle.apps.fnd.appcore.attachments.Model > oracle.apps.fnd.appcore.attachments.uiModel.view > AttachmentsVO. In the Data Model (right column), select the Attachments View. Shuttle the AttachmentsRenditionVO View Link over to the right column.

f. Save your changes.

---

**Note:** To enable the Flexfield at a later point in time it is recommended that the Attachments View Link be edited and selected during that wizard flow.

---

### What Happens When You Create an Attachment View Link

After you have completed all the steps in the wizard by clicking Finish to create your Attachment view link:

- The Attachment view link is created, using the Name and Package provided in Step 2.

- If the view link was selected to display only a subset of the categories in Step 6. A custom property (OAF_ATTACHMENT_CATEGORY) is created on the Attachments view link, with a comma separated list of the categories that you selected in Step 6 of the wizard. If the decision was to show all categories. The custom property will still be created, but with an empty list.

- The AttachmentEntityName transient attribute is created on the Entity view object. Its value is the entity name that you entered in Step 4 of the wizard.

---

**Note:** This transient attribute should not be included in your UI.

- An Accessor is created on the Entity view object, using the View Link Accesor Name supplied in Step 3.

- New Attachment Entity information (Step 4) is stored to the FND_DOCUMENT_ENTITIES and FND_DOCUMENT_ENTITIES_TL tables.

- All new or changed category information (Step 5) is stored to the FND_DOCUMENT_CATEGORIES and FND_DOCUMENT_CATEGORIES_TL tables.

- All new or changed mapping information (Step 5) between the Entity Name and the Categories is stored in the FND_DOC_CATEGORIES_TO_ENTITIES table.

- New Entries defined will be added to the Data Model of the Application Module selected in Step 7.

The WHERE clause that links the Entity view object to the Attachments view object is derived based on the selected entity primary key columns. For example, if the entity
primary key for the FND_DEMO_DEPT entity is made up of the column DEPT_NUM then the view link query would be as follows:

Example Source WHERE Clause

{(:Bind_DeptNum = PK1_VALUE) AND
 (:Bind_AttachmentEntityName = ENTITY_NAME)}

Example Destination WHERE Clause

{(:Bind_Pk1Value = FndDemoDeptEO.DEPT_NUM) AND
 (:Bind_EntityName = 'FND_DEMO_DEPT')

where FND_DEMO_DEPT is the value that you entered in the Entity Name field in Step 4 of the wizard.

To Edit an attachment view link:

1. In JDeveloper, choose Application Navigator > Model Project. Right-click and choose New from the menu to open the New Gallery.

2. Choose the Business Tier > ADF Business Components category. Select the Attachment View Link item, as shown in Figure 3.

3. Click OK to access Step 1 of the Attachment View Link wizard, as shown here.
4. Select **Edit View Link**, click **Next** to move to Step 2 View Link, as shown here. Select the attachment view link to edit from the available list.
5. Click Next to access Step 3. The Name dialog will appear.

**Note:** The steps from this point mirror their similarly named counterparts in the Edit flow. The rest of this section will focus on what modifications are allowed.

**Note:** The edits to each step are currently independent. For example, changing the Entity Name will not move the categories mapped to the old Entity Name to the new Entity Name. This will still need to be performed on subsequent steps.
6. Step 3 – Name. The Module can be changed at this step. The new module will be used in any queries against stored data. Any changes to Entity Name or Categories will be stored using the module.


**Note:** Changing the View Link Accessor Name currently has no effect.

8. Step 5 – Attachment Entity. Create a New Entity, or select a different existing Entity.

9. Step 6 – Category Mappings. Create Categories, Add existing Categories, Modify mapped Categories, Remove Categories from being mapped to the Entity Name.

10. Step 7 – Categories. Switch whether the View Link is to Display All Categories, manage which Categories to show if the View Link isn’t displaying all.

11. Step 8 – Application Module. After selecting the Application Module, the table will show the existing usages for the View Link in the Application Module Data Model. Use **Add** to create additional View Link Usages, or **Remove** to delete any existing View Link Usages.

**Selecting from the Content Server Configuration Step**

Setting the RefreshCondition property is only required when the repositoryMode property is set to true on the Attachment component. Setting this property for the taskflow allows the Document Picker to be used more than once within the Attachments table or popup in the same session.

1. Go to the Bindings tab for that page.
2. Click the link to the Page Definition File, which is located at the top of the page.
3. Select the attachmentRepositoryBrowseTaskFlow1 task flow from the list of Executables.
4. Go to the Property Inspector and set the RefreshCondition property based on the component in use. For the Attachments and Attachments Carousel components, use #{AttachmentsTaskflowListener.refreshTaskflow}. For Attachment Views and Attachment Link components, use #{AttachmentsV2TaskflowListener.refreshTaskflow}.
5. Save your changes.

**Update Page Bindings (recommended)**

Page Bindings need to be manually created to include AttachmentsExtraAttributes, AttachmentsRendition and Attachments DFF. They are not essential to the operation of any of the Attachment components. They are required for the Details Pane of the Attachments View component to show More Information or Renditions sections.

**Note:** Ensure that the Update Application Module steps have already been completed first. Otherwise, there will be no child entries available in Step 3. This step can be completed by using the Edit View Link option in the Attachments View Link Wizard in Step 8 – Application Module. Manual instructions are included in step 14 of To Create an Attachment View Link.
Reviewing the Bindings tab of a page will indicate whether the following steps are necessary. Check for entries in the Executables list starting with AttachmentsExtraAttributes and AttachmentsRenditions.

1. Open the page that contains the Attachment Views component.
2. In the Data Controls section of JDeveloper find the Attachments Data Control entry which was used to create the Attachment Views component. (Step 2 of To create an Attachment Views).
3. Expand the Attachments entry, drag the AttachmentsExtraAttribute child onto the page and choose a Read-Only table with all columns.
4. Remove the af:table just created from the page.
5. Back in the Data Controls drag the AttachmentsRendition child onto the page and choose a Read-Only table with all columns.
6. Remove the af:table just created from the page.
7. Back in the Data Controls, drag the AttachedDocumentsDFF child onto the page and choose Applications > Oracle Descriptive Flexfield.
8. Remove the fnd:descriptiveFlexfield just created from the page.
9. Open the page definition of the Page and go to the Source.
   Add the following source inside the <tree> node for the Attachments definition:
   ```xml
   <Properties>
   <CustomProperties>
   <Property Name="ExtraAttributes" Value="<ExtraAttributesId>" />
   <Property Name="Renditions" Value="<RenditionTreeId>" />
   <Property Name="DFF" Value="<DffIteratorId>" />
   </CustomProperties>
   </Properties>
   ```
10. Replace <ExtraAttributesId> with the Id of the tree node for the Attachments Extra Attribute that was created in step 3.
11. Replace <RenditionTreeId> with the Id of the tree node for the Attachments Rendition that was created in step 5.
12. Replace <DffIteratorId> with the Id of the iterator node for the Attachments DFF that was created in step 7.
13. Save the page definition.

   The Attachments View component will detect these exist properties have been configured and will start to use them.

**About Attachments Descriptive Flexfield**

The Attachment Views component has the ability to display additional custom metadata in the details pane using a descriptive flexfield. There is a new flexfield definition, FND_ATTACHED/Documents_DFF, which can be customized to allow additional custom metadata values to be entered and stored with each individual attachment. The structure of the descriptive flexfield can be configured using instructions from the Descriptive Flexfields chapter. The Attachments descriptive flexfield is context sensitive to the Attachments Category. Any change to the category in the details panel triggers the panel to refresh with the updated flexfield segments.
displayed. By default, the Attachment Views component does not show the descriptive flexfield, unless the page bindings have been updated. The flexfield is shown in the More Information section of the Attachment Views detail panel, if the correct view links and bindings have been created.

**Note:** There is a limitation to the use of the attachments descriptive flexfield. Only one instance of an attachment descriptive flexfield can be within an application module. This instance must be a child of a single attachments view object instance, and the descriptive flexfield instance must be named AttachedDocumentsDFF1.

**Note:** Most environments will have the Attachment DFF automatically patched. Some standalone environments may require the DFF to be manually deployed. A manual deployment is required where the following message is displayed:

```
Definition
oracle.apps.flex.fnd.applcore.attachments.model.view.AttachedDocumentsDFFVO
of type View Definition is not found.
```

Sign on to Fusion Applications and perform the following:

1. Navigate to Setup And Maintenance
2. Search for task Manage Applications Core Descriptive Flexfields
3. Perform a Blind Search
4. Click Attached Documents DFF
5. Click the Deploy Flexfield button

See Deploying Flexfields in a Standalone WebLogic Server Environment for more details.

**Custom Actions**

Attachment Views provides the capability of overriding the default items rendered in the List, Card and Carousel views with your own. A facet is provided for each View for product teams to develop their own item. The actions and any custom actions for the item will still be included in the same spot. To activate, set the value of the Rendering Property to “custom”.

<table>
<thead>
<tr>
<th>View</th>
<th>Facet</th>
<th>Rendering Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>List</td>
<td>customListItem</td>
<td>listItemViewer</td>
</tr>
<tr>
<td>Card</td>
<td>customCardItem</td>
<td>cardItemViewer</td>
</tr>
<tr>
<td>Carousel</td>
<td>customCarouselItem</td>
<td>carouselItemViewer</td>
</tr>
</tbody>
</table>
19.2.4.1 How to Create an Attachment View Link Without Selecting an Application Module

If you chose to keep the Application Module option unselected in Step 10, you need to add your newly created attachments view link to your application module data model as follows:

1. Choose Application Navigator > AM name. Right-click and choose to open the Application Module in the editor.

2. Choose Data Model from the list of categories to open the Data Model Components dialog.

3. Select the newly created Attachment View Link located in the left column and select the view object that you created the view link for in the right column. Shuttle the Attachment View Link over to the right column.

4. Save your changes.

19.2.4.2 How to Update the Application Module

Dragging and dropping the Attachments collection automatically adds the Attachments collection as a page binding. These page bindings are required to make Attachment views function. There are additional child collections that also must be added to the page binding to provide support for renditions in the "More Information" and "Renditions" sections of the Details pane. To make the child collections available, you must add them to the Application Module's data model.

To add the child collections to the Application Module, do the following:

1. In the Application Navigator, find the Application Module in the Model project in which the Attachment View Link for the attachment views component is found. Right-click and choose to open the Application Module in the editor.

2. Select Data Model from the list of categories to open the Data Model Components dialog.

3. From the Available View Objects list, go to oracle.apps.model.Model > oracle.apps.fnd.applcore.attachments.Model > oracle.apps.fnd.applcore.attachments.uiModel.view > AttachmentsVO and select AttachmentsExtraAttributeVO.

4. In the Data Model list, select the attachments view. Then, shuttle the AttachmentsExtraAttribute view link over to the right column.

5. Repeat this process for any other attachment views in the data model that must be included the extra attributes.

6. Repeat Steps 4 and 5 for the AttachmentsRenditionVO, which is located in the same package as AttachmentsExtraAttributeVO.

7. Save your changes.

19.2.5 What Happens When You Create an Attachment View Link

After you have completed all the steps in the wizard by clicking Finish to create your Attachment view link:
• The Attachment view link is created, including generating the Accessor in the source view object.

• All new or changed category information is stored to the FND_DOCUMENT_CATEGORIES and FND_DOCUMENT_CATEGORIES_TL tables.

• A custom property (OAF_ATTACHMENT_CATEGORY) is created on the Attachments view link, which is a list of all the categories that you select in Step 4 of the wizard.

  The custom property stores an empty list when you select the Display All Available Categories checkbox. When this checkbox is not selected, the custom property stores a comma-separated list of categories (using the CATEGORY_NAME from the FND_DOCUMENT_CATEGORIES table) that you manually selected from the list of available categories.

• The AttachmentEntityName transient attribute is created on the Entity view object. Its value is the entity name that you entered in Step 3 of the wizard.

  Note:
  This transient attribute should not be included in your UI.

• New Attachment Entity information is stored to the FND_DOCUMENT_ENTITIES and FND_DOCUMENT_ENTITIES_TL tables.

  The WHERE clause that links the Entity view object to the Attachments view object is derived based on the selected entity primary key columns. For example, if the entity primary key for the PO_INVOICES entity is made up of the columns INVOICE_HEADER_ID and INVOICE_LINE_ID then the view link query would be as follows:

  where PO_INVOICES is the value that you entered in the Entity Name field in Step 3 of the wizard.

  Example 19-2  Source WHERE Clause

  (:Bind_InvoiceHeaderId = PK1_VALUE) AND
  (:Bind_InvoiceLineId = PK2_VALUE) AND
  (:Bind_AttachmentEntityName = ENTITY_NAME)

  Example 19-3  Destination WHERE Clause

  (:Bind_Pk1Value = PoInvoiceLines.INVOICE_HEADER_ID) AND
  (:Bind_Pk2Value = PoInvoiceLines.INVOICE_LINE_ID) AND
  (:Bind_EntityName = PO_INVOICES)

19.2.6 About Deleting the Business Object

Deleting the business object from the database does not automatically apply to any of its Attachments. Subsequently, if your business functionality calls for the deletion of the business object (as opposed to a programmatic deletion, where the record is flagged as deleted but is kept around for auditing purposes), you also must delete all of its Attachments. Otherwise, these records will continue to persist in the Attachments tables. If another business object for your entity is created with the same primary key values, these Attachments will immediately be attached to the new record.
What you do depends on the method or methods you used to delete the business object in your own functionality. The programmatic method to delete the attachments will be the same. Typically, overriding the `remove()` method of the business object view object will allow you to programmatically access the Attachments detail collection via the View Link Accessor. When you have access to this collection, you can loop through calling the `remove()` method of the collection, shown in Example 19-4.

**Example 19-4 remove() Method**

```java
public void remove() {
    Row currRow = this.getCurrentRow();
    RowSet attachedItems = (RowSet) currRow.getAttribute("Attachments1");
    while (attachedItems.hasNext()) {
        Row currAttachment = attachedItems.next();
        currAttachment.remove();
    }
    super.remove();
}
```

### 19.2.7 About Assigning Categories to the Attachment Entity

It is necessary to assign one or more attachment categories to your attachment entity. The "Manage Attachment Categories" setup UI can be used to create and update the relationships between your categories and entities. You can search for a particular attachment category, then search and select the attachment entities to assign to the category. For more information, see About Integrating Attachments Task Flows into .

Relationships between categories and entities can also be maintained using the Manage Attachment Entities setup UI.

The Entity-Category relationships are stored in the `FND_DOC_CATEGORIES_TO_ENTITIES` table. There is a many-to-many relationship between Entities and Categories, allowing one category to be assigned to multiple Entities.

The relationships stored in this table are striped by `MODULE_ID` for seeding purposes, and the seed data loader can be used to seed this data for your product.

---

**Note:**

It is important to ensure that you seed your attachment entities and attachment categories before you seed the data in the `FND_DOC_CATEGORIES_TO_ENTITIES` table when extracting and uploading the data.

---

See the "Available Seed Data Loaders" table in Chapter 65 for information about Attachments Seed Data Loaders.

### 19.2.8 How to Create an Attachments Field, Attachments Table, Attachments Carousel, or Attachment Views

This section describes how to create an Attachments field, table, carousel, or views.

1. Create a page, page segment, or an applications panel that is bound to the attachments-enabled Master Data Collection (your business object data collection).
2. Drag the Detail Data Collection (the Attachments data collection) onto a databound drop target. The Create context menu appears.

3. Select Applications > Attachments to display the Create Attachments dialog, as shown in Figure 19-25.

![Create Attachments](image)

Choose Attachments Field, Attachments Table, Attachments Carousel, or Attachment Views. Click OK to display the Edit Taskflow Binding dialog.

The ConnectionName parameter is automatically set to 
#(AttachmentsTaskflowListener.connectionName) on the attachmentRepositoryBrowseTaskFow1 taskflow in the page bindings for your page. This parameter is used to derive the connection name of the content server connection to be used when the Document Picker taskflow is used in the Attachments UI at runtime.

4. Click OK. Do not make any changes on the Edit Taskflow Binding dialog.

Selecting Attachments Field or Attachments Table will add the Attachments component to the page. The mode will be set automatically to link or table. Selecting Attachments Carousel will add the Attachments Carousel component to the page. Selecting Attachment Views will add the Attachment Views component to the page. The selected component will be bound to the Detail Data Collection.

**Example 19-5 Adding Event IDs to the PartialTriggers**

```af
c<af:panelLabelAndMessage label="Attachment" partialTriggers="btnNext btnPrevious">
  c<fnd:attachment mode="link" repositoryMode="true" attachmentModel="#{bindings.Attachments1}"/>
</af:panelLabelAndMessage>
```

When using attachments in link mode, the Attachment component must be displayed inside a panelLabelAndMessage component and a partialTrigger must be added as an attribute. You must then add the IDs for the appropriate events on the page to the partialTriggers that cause the underlying business object record to change, which in turn requires the Attachments data to change. For example, the navigation buttons Next and Previous as shown here:

**19.2.9 What Happens When You Implement Attachments**

The following are examples of the generated source code:

**Example 19-6 Source Code for an Attachment Field**

```fnd
c<fnd:attachment mode="link" id="a1" attachmentModel="#{bindings.AttachmentsIterator}"/>
```
Example 19-7  Source Code for an Attachment Table

```xml
<fnd:attachment mode="table" id="a1"
    attachmentModel="#{bindings.AttachmentsIterator}"/>
```

Example 19-8  Source Code for an Attachments Carousel

```xml
<fnd:AttachmentsCarousel id="acl"
    attachmentModel="#{bindings.AttachmentsIterator}"/>
```

Example 19-9  Source Code for an Attachment View

```xml
<fnd:attachmentView id="aView"
    attachmentModel="#{bindings.AttachmentsIterator}"/>
```

19.2.10 How to Create an Attachments Column in an Applications Table

You can create an attachments column in an applications table by dragging a Master Data Collection (your business object data collection) onto your drop target.

---

**Note:**

- The `mode` attribute is automatically set to `columnLink` on the Attachment component when creating an attachment column.
- The bindings, shown in Example 19-10, only show when the Attachments are bound to a data control.
- In `columnLink` mode, the `columnModel` attribute also must be set with the value of the Attachments Accessor.

---

To create an attachments column when creating your applications table, do the following:

1. Drag your Master Data Collection (your business object data collection) onto your drop target (Page, Page Segment, or Applications Panel). The Create Context Menu is displayed.
2. Select Create, Applications Table to open the Applications Table wizard, Configure Table Patterns dialog.
3. Select the Attachments option.
4. Click OK. Do not make any changes on the Edit Taskflow Binding dialog.
   
   An Applications Table is created with an Attachments column located in the rightmost position.
5. Manually add the `columnLinkTableModel` attribute to the attachments component. This attribute must be set to the value that identifies the Master Data Collection Model and must match the value set for the value property of the `af:table` component that contains the Attachment column. For example:

   ```xml
columnLinkTableModel="#{bindings.AuEmployee1.collectionModel}"
```
19.2.11 How to Create an Attachments Column in an Existing Applications Table

You can create an attachments column in an existing applications table by dragging a Detail Master Collection (the Attachments data collection) onto your drop target.

1. Drag your Detail Data Collection (the Attachments data collection) onto your drop target (af:table within the table facet of the Applications table). The Create Context Menu is displayed.

2. Select Create, Applications, Attachments Column. The Edit Taskflow Binding dialog is displayed.

3. Click OK. Do not make any changes on the Edit Taskflow Binding dialog.

4. Manually add the columnLinkTableModel attribute to the attachments component. This attribute must be set to the value that identifies the Master Data Collection Model and must match the value set for the value property of the af:table component that contains the Attachment column. For example:

   ```
columnLinkTableModel="#{bindings.AuEmployee1.collectionModel}"
```

Example 19-10  Source Code for an Attachment Column

```
<af:column sortable="false"
   headerText="#{applcoreBundle.ATTACHMENTS_COLUMN_HEADER}"
   width="200">
  <fnd:attachment mode="columnLink"
     attachmentModel="#{bindings['Attachments1']}"
     columnModel="#{row.attachments}"/>
  columnLinkTableModel="#{bindings.AuEmployee1.collectionModel}"
</af:column>
```

19.2.12 About Setting Up Required Properties

To implement your attachments successfully you must set the following required properties:

- usesUpload for a standard JSPX page
- usesUpload on the UI Shell

Note:

usesUpload must be set using either of the above methods when uptaking either a taskflow or component that contains the attachments component.
Ensure that your uptake document includes this instruction.

This property serves as a reminder for anyone including Attachments as part of an "object" being provided that the destination JSPX page or UIShell must have the same settings so that the users can upload.

- RefreshCondition for the task flow

19.2.12.1 About Setting the usesUpload Property for a Standard JSPX Page

To be able to upload files from the desktop to the Content Server, you must set the usesUpload property to true.
For a standard JSPX page, you must set the `usesUpload` property on the `af:form` that contains your attachment component. For example:

```<af:form usesUpload="true">```

**19.2.12.2 Set the `usesUpload` Property on the UI Shell**

To set the `usesUpload` property value on the UI Shell's `af:form`, you need to set the Form Uses Upload property on the `itemNode` that represents the JSPX in the appropriate menu XML file.

1. Go to the Application Navigator and select the menu XML file.
2. Go to the Structure pane and select the item node representing the JSPX page.
3. Go to the Property Inspector > Advanced Properties > Page tab and set the Form Uses Upload property to `true`.
4. Save your changes.

**19.2.12.3 Set the `RefreshCondition` Property for the Taskflow**

Setting the `RefreshCondition` property is only required when the `repositoryMode` property is set to `true` on the Attachment component. Setting this property for the taskflow allows the Document Picker to be used more than once within the Attachments table or popup in the same session.

1. Go to the Bindings tab for that page.
2. Click on the link to the Page Definition File, which is located at the top of the page.
3. Select the `attachmentRepositoryBrowseTaskFlow1` task flow from the list of Executables.
4. Go to the Property Inspector and set the `RefreshCondition` property to `#(AttachmentsTaskflowListener.refreshTaskflow)`.
5. Save your changes.

**19.2.13 What Happens at Runtime**

Users have the ability to associate a URL, desktop file, repository file or folder, or text with a business object, such as an expense report, contract, or purchase order.

---

**Note:**

The ability to associate repository files or folders is available when the `repositoryMode` property is set to `true`.

---

When a user chooses to attach a repository file or folder, they are presented with the Document Picker dialog from which they can select one or more repository files or folders. After a repository file or folder has been attached, the type is displayed as either File or Folder.

The ability for a user to add, update, view, or delete an attachment is programmatically controlled using the `addAllowed`, `updateAllowed`, `deleteAllowed`,...
viewAllowed, deleteAllowed, and ReadOnly attributes on the Attachment component.

The ability to control what type of attachments can be added in the Attachments UI is controlled by the following attributes on the Attachment component: fileTypeEnabled, textTypeEnabled, urlTypeEnabled, and attachFolderAllowed.

The type of attachment that is used determines how the attachment is displayed when the user clicks the attachment link. The browser and client configuration determines what desktop application is used to display the attachment. When clicking on the link to access the content of a File or Text type Attachment, the user will be prompted to save the file. For their protection, users should be encouraged not to open files inside the browser.

Repository files are viewed in the same way as desktop files. However, if you click the link in the File Name/URL column in the Attachments table for an attached folder, a browser window/tab opens and displays the list of files and folders within the attached folder.

19.2.14 How to Update Page Bindings for the Attachment Views Component

Page bindings must be manually created to include AttachmentsExtraAttributes and AttachmentsRendition. They are not required for the operation of any of the Attachment components. However, in order to display the "More Information" and "Renditions" sections in the Details pane of the Attachments View component, page bindings are required.

**Note:**

- Review the **Bindings** tab of a page to determine if the following procedure is necessary. Check for entries in the list of executables, starting with AttachmentsExtraAttributes and AttachmentsRenditions.

- Before you update page bindings, ensure that you have updated the Application Module so that child entries will be available when you perform Step 3 in the procedure that follows. For more information, see How to Update the Application Module.

To update page bindings, do the following:

1. Open up the page that contains the Attachment Views component.

2. In the Data Controls section of Oracle JDeveloper, find the Attachments Data Control entry that was used to create the Attachment Views component.

3. Expand the Attachments entry, drag the AttachmentsExtraAttribute child onto the page, and select a read-only table with all columns.

4. Remove the **af:table** just created from the page.

5. In the Data Controls section, drag the AttachmentsRendition child onto the page and choose a Read-Only table with all columns.

6. Remove the **af:table** just created from the page.
7. In the Data Controls, drag the AttachedDocumentsDFF child onto the page and choose **Applications > Oracle Descriptive Flexfield**.

8. Remove the `fnd:descriptiveFlexfield` just created from the page.

9. Open the page definition of the page and go to the source. Add the following source inside the tree node for the Attachments definition:

   ```xml
   <Properties>
   <CustomProperties>
     <Property Name="ExtraAttributes" Value="<ExtraAttributesId>" />
     <Property Name="Renditions" Value="RenditionTreeId" />
     <Property Name="DFF" Value="<DffIteratorId>" />
   </CustomProperties>
   </Properties>
   ```

10. Replace `ExtraAttributesId` with the Id of the tree node for the Attachments extra attribute that was created in Step 3.

11. Replace `RenditionTreeId` with the Id of the tree node for the Attachments Rendition that was created in Step 5.

12. Replace `<DffIteratorId>` with the Id of the iterator node for the Attachments DFF that was created in step 7.

13. Save the page definition.

   The Attachments View component will detect that these properties have been configured and will start using them.

### 19.3 About Displaying Attachments for Multiple Entities in the Same Table

For a user interface that shows information for more than one business object, the Attachments component can be configured to display the attachments for two or more of these business objects.

For example, when opening up the Attachments from the Department section of your UI page, you might want to display attachments for all Employees that exist in that Department. This example will be used throughout this section to help clarify the instructions.

This configuration is only possible when the underlying view object for your page contains the primary keys for each of the business objects whose attachments you want to show in your page. For example, the `DeptEmpVO` would need to include attributes for both the Department primary key and the Employee foreign key. In this example, the Department is considered to be the primary business object and the Employee is the secondary business object. The example uses only two business objects, but it is possible to display the attachments for more than two business objects using this feature.

Your primary business object is the one that you linked to the AttachmentsVO at design time, when you created your Attachment view link. Attachments can only be added to the primary business object when using this feature.

The `actionEntity` property can be used to set the primary business object and controls which attachments can be updated and deleted. When set, any document attached to a different entity is displayed for reference purposes only. It cannot be updated or deleted.
19.3.1 About Configuring the Attachments Component to Display Attachments for Multiple Entities

This section provides details on how to configure your data model to display attachments for two or more business objects. These instructions assume that you have already created the Attachments view link between one business object view object and the Attachments view object, as described in How to Create Attachment View Links.

19.3.1.1 How to Configure the Attachments Component to Display Attachments for Two or More Business Objects

Do the following:

1. Open the page that contains the Attachment component that is bound to the ViewLink modified above.
2. Select the Attachment component.
3. Set the `actionEntity` property to the `ENTITY_NAME` of the primary business object. By setting this property, you are allowing users to update and delete attachments belonging to the primary business object whilst disabling the ability for users to update and delete attachments for the secondary and all subsequent business objects.

19.3.1.2 How to Configure the Data Model to Display Attachments for Two or More Business Objects

Do the following:

1. Open the view object for your business object.
2. Navigate to the Attributes tab. Add a new transient attribute for the secondary business object for which you want to display attachments. Use the existing `AttachmentEntityName` attribute as an example, naming the new attribute `AttachmentEntityName1`.
3. Change the Expression value in the View Attribute screen to be the Attachment `ENTITY_NAME` as defined for your business object in the `FND_DOCUMENT_ENTITIES` and `FND_DOCUMENT_ENTITIES_TL` tables.
4. Repeat Steps 2 and 3 for any subsequent business objects for which you want to show attachments. Make sure that you modify the attribute name and expression value for each.
5. Close the view object.
6. Open the attachment view link.
7. Link your secondary business object to the Attachments view object.
   a. Navigate to the Relationship tab and click the Attributes Edit icon.
   b. Select the foreign key reference column in your view object from the Source Attribute list and then select the corresponding `PknValue` column in the Attachments view object (starting with `PknValue`) from the Destination Attribute list. Click Add to add this pair.
Repeat this for each column that makes up the foreign key, which identifies your secondary business object.

c. Select the `EntityName` attribute (created in Step 2) in the Attachments view object from the Destination Attribute list. Click Add to add this pair.
Repeat this for each of the AttachmentEntityName transient attributes.

d. Navigate to the Query tab, and click the Source Edit icon.

e. Modify the auto-generated Source and Destination queries to match the appropriate Entity Names and Primary Key values. Put parentheses around the criteria that pertain to one business object and change the AND to an OR in between each business object. Make sure your parentheses are matched correctly, as shown in examples Example 19-11 and Example 19-12.

7. Navigate to the Source view for the view link and edit the source XML to include the extra attributes in the Destination array. Make sure you include attributes for each additional business object for which you want to display attachments.

The AttrArray for the destEnd ViewLinkDefEnd must map to the AttrArray for the sourceEnd ViewLinkDefEnd.

The design time code inserts an array of unique items where the runtime code needs a matching item list, as shown in Example 19-13. (The added items are in bold).

8. Navigate back to the General tab and expand the Custom Properties section. Modify the value of the OAF_ATTACHMENT_CATEGORY custom property to control the attachments that are displayed in your page, based on category and to control the Category droplist values.

Note:
This can be further controlled using data security.

9. Save all your changes.

Example 19-11 Source Query

```
((:Bind_EmployeeId = PK1_VALUE) AND (:Bind_AttachmentEntityName = ENTITY_NAME))
```

```
OR
```

```
((:Bind_DeptNum = PK1_VALUE) AND (:Bind_AttachmentEntityName1 = ENTITY_NAME))
```

Example 19-12 Destination Query

```
((:Bind_Pk1Value = FndDemoDeptEmpEO.EMPLOYEE_ID) AND (:Bind_EntityName = 'FND_DEMO_EMP'))
```

```
OR
```

```
((:Bind_Pk1Value = FndDemoDeptEmpEO.DEPT_NUM) AND (:Bind_EntityName = 'FND_DEMO_DEPT'))
```

Example 19-13 Source and Destination Attribute Arrays

```
<AttrArray Name="Attributes">
```

```
<AttrArray Name="Attributes">
```
19.4 About Configuring Attachments Component Properties

You can use properties to change the default behavior of the Attachments, Attachments Carousel, and Attachment Views components user interfaces.

For a complete list of these properties, see Configuring Attachments Component Properties.

19.4.1 About Using FUSE

Support has been included for using Attachments in pages built for Oracle Fusion Applications User Simplified Experience (FUSE). When using Attachments in a FUSE page, you will need to adhere to the following rules in order to implement a FUSE compliant design:

<table>
<thead>
<tr>
<th>Table 19-1</th>
<th>FUSE Implementation Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule</td>
<td>Description</td>
</tr>
<tr>
<td>fuse</td>
<td>Boolean: true / false. True renders the attachment component for a FUSE UI page. Only link mode is supported for FUSE.</td>
</tr>
<tr>
<td>mode=&quot;link&quot;</td>
<td>Link mode is the only supported mode of operation.</td>
</tr>
<tr>
<td>repositoryMode=&quot;false&quot;</td>
<td>Only Attachments of type File or URL are allowed. The Repository File/Folder and Text attachment types need to be disabled.</td>
</tr>
<tr>
<td>textTypeEnabled=&quot;false&quot;</td>
<td>You will not be prompted for confirmation before an Attachment is deleted.</td>
</tr>
<tr>
<td>confirmDelete=&quot;false&quot;</td>
<td>String. Value selected as the default in the Category poplist when adding new attachments.</td>
</tr>
</tbody>
</table>

The following properties are not used when in FUSE mode:

- ApprovalEnabled
About Working with Attachments Programmatically

19.5 About Working with Attachments Programmatically

To support headless or automated processing of Attachments, methods have been provided in the
oracle.apps.fnd.appcore.attachments.attachmentService.applicationModule.AttachmentServiceAM class.

Before you begin:

- If the Application or Project has not already been configured to use Attachments, follow the instructions in Performing Prerequisite Tasks.
- Find or create an instance of the AttachmentServiceAM.

Working with streams:

The Attachment APIs use the stream type for receiving and sending the actual files. When implementing your designs, keep the following rules in mind to avoid common issues that can occur.

- Ensure that Response Streams are always closed (is.close()) in a finally block.
- Ensure responses are processed immediately in full, and in order, prior to beginning the next request in the context of a thread. A delay in processing might result in a time out.
An example of this would be when downloading a stream that contains a set of instructions that are processed from the stream one line at time with each requiring several seconds before the next line is processed. With enough instructions lines, this might cause the stream to time out. The better approach would be to process the stream and store the instructions in another memory structure.

- Do not attempt to work with the Attachment API with two steams open in the context of a thread.

**Example 19-15  AttachmentServiceAM**

```java
// Enable the Attachments Service AM
AttachmentServiceAMImpl attachmentServiceAMImpl =
    (AttachmentServiceAMImpl) am.findApplicationModule("attachmentsServiceAM");
if (attachmentServiceAMImpl == null)
{
    attachmentServiceAMImpl =
    (AttachmentServiceAMImpl)am.createApplicationModule("attachmentsServiceAM",
        "oracle.apps.fnd.applcore.attachments.attachmentService.applicationModule.
            AttachmentServiceAM");
}
```

### 19.5.1 About Creating New Attachment Types

Methods have been provided for programmatically creating the File, Text and URL types. There is also an additional type, ManagedURL.

A managed URL attachment is intended for use with pages that are hosted on internal servers that are administered by Topology Manager. Since changes made by the Topology Manager might invalidate the scheme://domain:port portion of a URL resulting in a dead link, support for this variation of the URL type was created. To the end-user, this link will still appear as a URL. Internally, however, this Attachment will be created with a different Attachment type, TMURI.

Methods for attaching files that have been uploaded from a third-party application are also available. In this case, the developer has gotten the required information returned from uploading the file to complete the attachment. Where the file is already attached to another record, using the `copyAttachments` method to attach the file to the require record is recommended.

Table 19-2 summarizes the available methods for Attachments APIs.

**Table 19-2  Method Summary for Attachment APIs**

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
</table>
| void                | `attachFile(AttachmentsVORowImpl row, java.lang.String fileName,
               java.lang.String contentType, java.io.InputStream fileStream)` |
|                     | Updates the supplied attachments row with the information to attach a file. A new version is checked in if the file has already been uploaded. |
| AttachmentsVORowImpl | `attachFile(java.lang.String fileName, java.lang.String contentType,
               java.io.InputStream fileStream)` |
<p>|                     | Creates a new row from the internal attachments view object and populates the attributes to attach a file. |</p>
<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>attachManagedUrl(AttachmentsVORowImpl row, java.lang.String entAppShortName, java.net.URI uri)</td>
<td>For the given attachment row, it populates the attributes for the attachment type AttachmentsConstants.TMURI_TYPE.</td>
</tr>
<tr>
<td>AttachmentsVORowImpl</td>
<td>attachManagedUrl(java.lang.String entAppShortName, java.net.URI uri)</td>
<td>Creates a new row from the internal attachments view object and calls attachManagedUrl(AttachmentsVORowImpl row, String entAppShortName, URI uri) to populate.</td>
</tr>
<tr>
<td>void</td>
<td>attachText(AttachmentsVORowImpl row, java.lang.String text)</td>
<td>Updates the supplied attachments row with the information to attach text. A new version is checked in if the file has already been uploaded.</td>
</tr>
<tr>
<td>AttachmentsVORowImpl</td>
<td>attachText(java.lang.String text)</td>
<td>Creates a new row from the internal attachments view object and populates the attributes to attach text.</td>
</tr>
<tr>
<td>void</td>
<td>attachUrl(AttachmentsVORowImpl row, java.lang.String url)</td>
<td>Updates the supplied attachments row with the information to attach a static URL.</td>
</tr>
<tr>
<td>AttachmentsVORowImpl</td>
<td>attachUrl(java.lang.String url)</td>
<td>Creates a new row from the internal attachments view object and populates the attributes to attach URL.</td>
</tr>
<tr>
<td>void</td>
<td>setAttachmentSequence(AttachmentsVORowImpl row)</td>
<td>Sets the sequence number for the row that is passed in. Should be used where end user ordering has been enabled.</td>
</tr>
<tr>
<td>Void</td>
<td>attachRendition(AttachmentsVORowImpl attachmentRow, AttachmentsRenditionVORowImpl renditionRow, java.lang.String name, java.lang.String originalName, java.lang.String fileType, java.io.InputStream fileInputStream)</td>
<td>Upload a rendition to the Content Server.</td>
</tr>
</tbody>
</table>
Table 19-2  (Cont.) Method Summary for Attachment APIs

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttachmentsRendition</td>
<td>attachRendition(AttachmentsVORowImpl attachmentRow, java.lang.String name, java.lang.String originalName, java.lang.String fileType, java.io.InputStream fileInputStream)</td>
</tr>
<tr>
<td></td>
<td>Upload a rendition to the Content Server.</td>
</tr>
<tr>
<td>VORowImpl</td>
<td>attachRendition(AttachmentsVORowImpl attachmentRow, java.lang.String name, java.lang.String originalName, java.lang.String fileType, java.io.InputStream fileInputStream, java.lang.Long fileSize)</td>
</tr>
<tr>
<td></td>
<td>Upload a rendition to the Content Server.</td>
</tr>
</tbody>
</table>

When end-user ordering has been enabled in the view for the attachment component, any programmatically created rows for the entity must have their sequence value set as well. Calling the method `setAttachmentSequence`, passing in the newly created row, will set the correct sequence value.

All methods are overloaded, providing two options for creating a new Attachment. The first option creates the new Attachment row from the Attachment collection linked to the parent view object. This is illustrated in Example 19-16.

The second option is to allow the method to create the Attachment row at the same time, as shown in Example 19-17.

If you use the second option, it is essential for the Entity and Primary Key values to be set programmatically as well. Getting any of these values wrong may result in the Attachment being lost or assigned to the wrong record. While the second type looks like less work it would be a better practice to use the first approach for creating new Attachments.

**Note:**

Assigning an Attachment to the wrong record could allow an end-user to view a document they would not normally be able to view.

It will also be necessary to set the Category value for the new row before committing the transaction as shown in both examples. The value expected is the CATEGORY_NAME from the FND_DOCUMENT_CATEGORIES table. Since the value is not validated, use the following method to ensure that you are providing a category value that is mapped to your Document Entity Name:

```java
newAttachment.setCategoryName("MISC");
```

**Example 19-16  Code for Creating a New Attachment Row: Option 1**

```java
// Retrieve the department for adding the new Attachment row to
ViewObject vo = am.findViewObject("FndDemoDepl1");
vo.setWhereClause("DEPT_NUM = :deptNum");
vo.defineNamedWhereClauseParam("deptNum", null, null);
vo.setNamedWhereClauseParam("deptNum", departmentNumber);
vo.executeQuery();
Row currDepartment = vo.next();
```
// Retrieve the Attachments Collection
RowSet attachments = (RowSet) currDepartment.getAttribute("Attachments");
// Create new Attachment Row
AttachmentsVORowImpl newAttachment = (AttachmentsVORowImpl) attachments.createRow();
// Add Managed Link
attachmentServiceAMImpl.attachManagedUrl(newAttachment, textEntAppShortName, textURI);
// Set the Category Value
newAttachment.setCategoryName("MISC");

Example 19-17  Code for Creating a New Attachment Row: Option 2

// Add Managed Link
newAttachment =
attachmentServiceAMImpl.attachTopologyManagedUrl(textEntAppShortName, textURI);
newAttachment.setEntityName("FND_DEMO_DEPT");
newAttachment.setPk1Value(departmentNumber);
// Set the Category Value
newAttachment.setCategoryName("MISC");

19.5.2 About Retrieving Attachments

The InputStream and OutputStream methods have been supplied for retrieving Attachment types of FILE or TEXT. The URL or TMURI types are not stored on the content server. Their URL values can be retrieved with the getAttachmentUrl() method, which is described in Table 19-3.

Table 19-3  Methods for Retrieving Attachments

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.io.InputStream</td>
<td>getAttachmentInputStream(AttachmentsVORowImpl row)</td>
</tr>
<tr>
<td></td>
<td>This method is deprecated. Use</td>
</tr>
<tr>
<td></td>
<td>getInputStream(AttachmentsVORowImpl row) instead.</td>
</tr>
<tr>
<td>java.io.InputStream</td>
<td>getInputStream(AttachmentsVORowImpl row)</td>
</tr>
<tr>
<td>Retrieve the attachment content for the supplied attachment row as a stream.</td>
<td></td>
</tr>
<tr>
<td>Unlike</td>
<td>getAttachmentInputStream(AttachmentsVORowImpl row), which embedded an error message in InputStream, this method throws an exception if there is an error downloading the data from the content server.</td>
</tr>
<tr>
<td>java.io.OutputStream</td>
<td>getAttachmentStream(AttachmentsVORowImpl row)</td>
</tr>
<tr>
<td>Retrieve the attachment content for the supplied attachment row as a stream.</td>
<td></td>
</tr>
<tr>
<td>java.lang.String</td>
<td>getAttachmentUrl(AttachmentsVORowImpl row)</td>
</tr>
<tr>
<td>Retrieve URL for the attachment content for the supplied attachment.</td>
<td></td>
</tr>
</tbody>
</table>
Table 19-3 (Cont.) Methods for Retrieving Attachments

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.io.InputStream</td>
<td>getInputStream(AttachmentsVORo java.lang.String rendition, java.lang.Boolean requireRendition) pl1 row, throws java.io.IOException, RenditionNotFoundException</td>
<td>Retrieve the named rendition for the attachment content for the supplied attachment row as a stream. Rendition name is case sensitive. Setting requireRendition to true validates the availability of the rendition against the local rendition collection. If the rendition is not available, a RenditionNotFoundException is thrown.</td>
</tr>
</tbody>
</table>

**Note:**

InputStream and OutputStream provide a handle to a file on the Content Server. The connection to the Content Server can only support one Stream at a time. Ensure that you have finished all processing on the current Stream before starting the next. Otherwise you may experience a loss of data in the first Stream. If it is important to have two open Streams, create a second thread and open the second Stream there.

19.5.3 About Using Attachment Utilities

Two utility methods are provided enable a list of Attachments to be copied to another record as identified by the unique combination of the Document Entity Name and Primary Key values. The list is constructed of the Attachment rows from one or more other records.

The copyAttachments utility does not duplicate the files on the content server. When the method is complete, the destination record will point to the same files in the content server as the source record.

The duplicateAttachments utility does duplicate the files on the content server. When the method is complete, the destination record will point to the new content on the content server.

When constructing the list of Attachments, consider whether Category Data Security has been implemented (see About Attachment Category Data Security). If implemented, the list of Attachments available may not be the full list depending on the permissions granted to the current user. If the full list is required it will be necessary to bypass Category Data Security by setting the dataSecurityDisabled flag on AttachmentsVOImpl to true, as shown in Example 19-18.

**Note:**

If order is important, the list is processed in the order it is received. Based on the default ordering the UI, this would place the first attachment in the list last when viewed.
### Table 19-4  AttachmentsVORowImpl

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creates copies of the supplied attachment rows assigning them to the entity values as provided.</td>
</tr>
<tr>
<td></td>
<td>Creates copies of the supplied attachment rows assigning them to the entity values as provided. Content on the content server is also copied.</td>
</tr>
</tbody>
</table>

### Example 19-18  AttachmentsVOImpl

AttachmentsVOImpl attachmentVO = attachmentServiceAMImpl.getAttachments1();
attachmentVO.setDataSecurityDisabled(true);
// assign where clause conditions
attachmentVO.executeQuery;

### 19.5.4 About Managing Attachments HTTP Sessions

Generally, the Attachments component handles resources effectively; there is normally no need to manage resources programatically. However, if the Attachments component is holding a large number of HTTP sessions open for a long time, you might need to dispose of the unnecessary sessions. You can do this by calling the `dispose` method, which is described in Table 19-5.

### Table 19-5  Method Summary for Handling a Large Number of Sessions

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>public static void dispose()</td>
</tr>
<tr>
<td></td>
<td>Cleans up resources held by the Oracle WebCenter Content connection.</td>
</tr>
</tbody>
</table>

### 19.5.5 About Creating Third-Party Attachments

By default, the Attachment files on the Content Server are locked down. When a request is made using the Attachments functionality, it includes additional parameters that are processed on the Content Server. The user’s privileges are temporarily elevated so that the request can be performed.
Note:
These requests are always made in the security context of the application that owns the Attachment. Access to Attachments is through the View Link defined in the model. A user that is unable to access your business record cannot access that record’s Attachments. In using the Attachments programmatically it is important that the implementation does not circumvent this design.

Methods that support attaching files to business-object records using third-party applications are described in the table that follows.

### Table 19-6 Methods Supporting Third-Party Attachments

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.util.Map</td>
<td>signUpload(java.lang.String title)</td>
<td>Creates a digital signature to allow documents to be checked into the Attachments security group on the Content Server.</td>
</tr>
</tbody>
</table>

The process to create an Attachment using a third-party application is the following:

1. The third-party application makes a request to the business object to retrieve a signature for uploading a file with a provided title to a record in the business object.

2. The business object evaluates whether the requesting user is allowed to attach a file to the record.

3. The business object makes a call to the `signUpload` method, which returns an array of name value pairs that make up the signature.

4. The business object returns the array to the third-party application.

5. The third-party application creates the request to create the new file. The request includes the array of name value pairs returned from the business object.

6. The third-party application sends the request to the Content Server.

7. The response from the Content Server includes the document ID and version number of the new document.

8. The third-party application makes a request to the business object to create a new Attachment for the record using the new document ID and version number.
9. The business object calls the attachRepositoryFile method for its record to create the new Attachment record.

Since the attachRepositoryFile method validates that the user making the request to create the attachment is the same user that created the file on the Content Server, all of the steps in this process must be performed by the same user.

The service call made to create the file on the Content Server (Step 5) is the CHECKIN_UNIVERSAL service. For more information, see the Services Reference for Oracle WebCenter Content.

When setting parameters, you must use the guidelines described in the table that follows.

<table>
<thead>
<tr>
<th>Table 19-7 Service-Call Parameter Guidelines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>dDocAuthor</td>
</tr>
<tr>
<td>dDocTitle</td>
</tr>
<tr>
<td>dDocType</td>
</tr>
<tr>
<td>dSecurityGroup</td>
</tr>
<tr>
<td>dDocAccount</td>
</tr>
<tr>
<td>primaryFile</td>
</tr>
<tr>
<td>alternateFile</td>
</tr>
<tr>
<td>dDocName</td>
</tr>
</tbody>
</table>

19.5.6 About Implementing Attachments with PL/SQL

PL/SQL equivalents exist for some of the Java methods described in About Creating New Attachment Types and About Using Attachment Utilities. These PL/SQL operations appear in the following table.

<table>
<thead>
<tr>
<th>Note:</th>
</tr>
</thead>
<tbody>
<tr>
<td>There can be no PL/SQL equivalents of the Java APIs that interact with the Content Server. PL/SQL is limited to working with the information stored in the Application Database only. There are no options available to access or create files stored on the Content Server.</td>
</tr>
</tbody>
</table>
About Working with Attachments Programmatically

Table 19-8  PL/SQL Attachments Implementation

<table>
<thead>
<tr>
<th>Method</th>
<th>PL/SQL Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachManagedURL</td>
<td>PROCEDURE ATTACH_MANAGED_URL (</td>
</tr>
<tr>
<td></td>
<td>X_ENTITY_NAME in VARCHAR2,</td>
</tr>
<tr>
<td></td>
<td>X_PK1_VALUE in VARCHAR2,</td>
</tr>
<tr>
<td></td>
<td>X_PK2_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_PK3_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_PK4_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_PK5_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_CATEGORY_NAME in VARCHAR2,</td>
</tr>
<tr>
<td></td>
<td>X_ENT_APP_SHORT_NAME in VARCHAR2,</td>
</tr>
<tr>
<td></td>
<td>X_URI in VARCHAR2,</td>
</tr>
<tr>
<td></td>
<td>X_TITLE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_DESCRIPTION in VARCHAR2 DEFAULT NULL );</td>
</tr>
<tr>
<td>attachURL</td>
<td>PROCEDURE ATTACH_URL (</td>
</tr>
<tr>
<td></td>
<td>X_ENTITY_NAME in VARCHAR2,</td>
</tr>
<tr>
<td></td>
<td>X_PK1_VALUE in VARCHAR2,</td>
</tr>
<tr>
<td></td>
<td>X_PK2_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_PK3_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_PK4_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_PK5_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_CATEGORY_NAME in VARCHAR2,</td>
</tr>
<tr>
<td></td>
<td>X_URL in VARCHAR2,</td>
</tr>
<tr>
<td></td>
<td>X_TITLE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_DESCRIPTION in VARCHAR2 DEFAULT NULL );</td>
</tr>
<tr>
<td>copyAttachments</td>
<td>PROCEDURE COPY_ATTACHMENTS (</td>
</tr>
<tr>
<td></td>
<td>X_SRC_ENTITY_NAME in VARCHAR2,</td>
</tr>
<tr>
<td></td>
<td>X_SRC_PK1_VALUE in VARCHAR2,</td>
</tr>
<tr>
<td></td>
<td>X_SRC_PK2_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_SRC_PK3_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_SRC_PK4_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_SRC_PK5_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_SRC_CATEGORY_NAME in VARCHAR2,</td>
</tr>
<tr>
<td></td>
<td>X_DST_ENTITY_NAME in VARCHAR2,</td>
</tr>
<tr>
<td></td>
<td>X_DST_PK1_VALUE in VARCHAR2,</td>
</tr>
<tr>
<td></td>
<td>X_DST_PK2_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_DST_PK3_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_DST_PK4_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_DST_PK5_VALUE in VARCHAR2 DEFAULT NULL,</td>
</tr>
<tr>
<td></td>
<td>X_DST_CATEGORY_NAME in VARCHAR2 DEFAULT NULL );</td>
</tr>
</tbody>
</table>

Updates the supplied attachments row with the information to attach a static URL.

Creates copies of the supplied attachment rows assigning them to the entity values as provided. This utility does not copy the content on the server; it creates a copy of the reference to the content.
Table 19-8  (Cont.) PL/SQL Attachments Implementation

<table>
<thead>
<tr>
<th>Method</th>
<th>PL/SQL Implementation</th>
</tr>
</thead>
</table>
| copyAttachment | PROCEDURE COPY_ATTACHMENT (  
|              | X_ATTACHED_DOCUMENT_ID in VARCHAR2,  
|              | X_ENTITY_NAME in VARCHAR2,  
|              | X_PK1_VALUE in VARCHAR2,  
|              | X_PK2_VALUE in VARCHAR2 DEFAULT NULL,  
|              | X_PK3_VALUE in VARCHAR2 DEFAULT NULL,  
|              | X_PK4_VALUE in VARCHAR2 DEFAULT NULL,  
|              | X_PK5_VALUE in VARCHAR2 DEFAULT NULL,  
|              | X_CATEGORY_NAME in VARCHAR2 DEFAULT NULL );  
|             | Creates a copy of the supplied attachment row. This utility does not copy the content on the server; it creates a copy of the reference to the content. |
| deleteAttachment | PROCEDURE DELETE_ATTACHMENT (  
|               | X_ATTACHED_DOCUMENT_ID in FND_ATTACHED_DOCUMENTS.ATTACHED_DOCUMENT_ID%TYPE );  
|               | Deletes an attachment from the database for the X_ATTACHMENT_DOCUMENT_ID. This does not delete the file in the content repository. |
| deleteAttachments | PROCEDURE DELETE_ATTACHMENTS (  
|               | X_ENTITY_NAME in FND_ATTACHED_DOCUMENTS.ENTITY_NAME%TYPE,  
|               | X_PK1_VALUE in FND_ATTACHED_DOCUMENTS.PK1_VALUE%TYPE,  
|               | X_PK2_VALUE in FND_ATTACHED_DOCUMENTS.PK2_VALUE%TYPE DEFAULT NULL,  
|               | X_PK3_VALUE in FND_ATTACHED_DOCUMENTS.PK3_VALUE%TYPE DEFAULT NULL,  
|               | X_PK4_VALUE in FND_ATTACHED_DOCUMENTS.PK4_VALUE%TYPE DEFAULT NULL,  
|               | X_PK5_VALUE in FND_ATTACHED_DOCUMENTS.PK5_VALUE%TYPE DEFAULT NULL );  
|               | Deletes all the attachments from the database for the specified parent record. This does not delete any files in the content repository. |

Note:

The Attachments procedures listed in About Creating New Attachment Types and About Using Attachment Utilities, but not listed in Table 19-8, cannot be implemented in PL/SQL.

A debugging utility is also included. When enabled, messages are written to the console. This feature is not enabled by default. At runtime the same messages will be written as log messages if logging is set to FINEST for the applcore.db.plsql.attachments.FND_ATTACHMENTS module.

To begin writing log messages to the console, use the `PROCEDURE START_DEBUG` command.

To stop writing log messages to the console, use the `PROCEDURE STOP_DEBUG` command.

Example 19-19 displays sample code for implementing Attachments with PL/SQL with debugging enabled.
Example 19-19 Implementing Attachments with PL/SQL

```
SET SERVEROUTPUT ON;
DECLARE
    BAD_ENTITY_NAME_EXCEPTION EXCEPTION;
    PRAGMA EXCEPTION_INIT(BAD_ENTITY_NAME_EXCEPTION, -20001);
    P_ENTITY_NAME VARCHAR2(40)  := 'FND_DEMO_DEPT1';
    P_PK1_VALUE   VARCHAR2(150) := '10';
BEGIN
    BEGIN
        FND_ATTACHMENTS.START_DEBUG;
        FND_ATTACHMENTS.ATTACH_URL (P_ENTITY_NAME,
                                      P_PK1_VALUE, NULL, NULL, NULL,
                                      'MISC',
                                      'http://www.oracle.com', 'Oracle Coorporation',
                                      NULL);
        FND_ATTACHMENTS.END_DEBUG;
    EXCEPTION
        WHEN BAD_ENTITY_NAME_EXCEPTION THEN
            DBMS_OUTPUT.PUT_LINE ('The entity name supplied, ' || P_ENTITY_NAME || ', is not a registered document entity.');
    END;
END;
```

19.6 About Setting Up Miscellaneous Attachments Features

Custom actions include the following:

- Custom actions for the attachments component
- Custom actions for the attachment views components
- Approvals
- Redlining

19.6.1 About Custom Actions for the Attachments Component

Four facets (placeholders) are provided on the attachments component for product teams to add toolbar buttons to the toolbar and actions to the Action menu of the attachments table.

The facet `tableAppsTableSecondaryToolbar` is provided to add toolbar buttons to the Attachments Table toolbar.

The facet `linkAppsTableSecondaryToolbar` is provided for Link mode.

**Note:**

When using the `tableAppsTableSecondaryToolbar` or `linkAppsTableSecondaryToolbar` facets, you must also set the `SecondaryToolbarRendered` property on the attachments component to true to expose this facet in the Attachments table.

The facet `tableAdditionalActionItems` is provided to add action items to the Action menu in the attachments table menus.

The facet `linkAdditionalActionItems` is provided for Link mode.
19.6.2 About Custom Actions for the Attachment Views and Attachment Link Component

Facets are provided on the Attachment Views component to add custom actions. The following facets are provided:

- **appsTableSecondaryToolbar**: adds toolbar buttons to the Attachment Views toolbar. Three buttons, Sel, All, and One, are shown in the following figure.

- **additionalActionItems**: adds action items to the Action menu in the Attachment Views menu. Five menu options, Show Selected Items, Show All Items, Show One Item, Selected Row Count, and All Row Count, are shown in the following figure.

![Custom Actions and Facets](image)

You can also use facets inline for adding icons or action items. These facets are incorporated into the actions section of the attachment, as shown in the previous figure. There is no central facet for the inline facets. As Oracle Application Development Framework rules do not allow this type of sharing, you must add each icon and item that must be identical for each view. This provides the flexibility for icons or actions to change from view to view for any implementation.

The rendering of the facets are controlled with properties. By default, the facets are not rendered.

<table>
<thead>
<tr>
<th>View</th>
<th>Facet</th>
<th>Rendering Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>additionalTableInlineActionItems</td>
<td>additionalTableInlineActionItemsRendered</td>
</tr>
<tr>
<td>Table</td>
<td>additionalTableInlineIconItems</td>
<td>additionalTableInlineIconItemsRendered</td>
</tr>
<tr>
<td>List</td>
<td>additionalListInlineIconItems</td>
<td>additionalListInlineIconItemsRendered</td>
</tr>
<tr>
<td>List</td>
<td>additionalListInlineActionItems</td>
<td>additionalListInlineActionItemsRendered</td>
</tr>
<tr>
<td>Card</td>
<td>additionalGridInlineActionItems</td>
<td>additionalGridInlineActionItemsRendered</td>
</tr>
<tr>
<td>Card</td>
<td>additionalGridInlineIconItems</td>
<td>additionalGridInlineIconItemsRendered</td>
</tr>
</tbody>
</table>

19.6.2.1 About Working with Custom Actions Programmatically

In support of developing the custom action functionality, utility methods are provided in the
oracle.apps.fnd.applcore.attachments.v2.uiPublic.utils.AttachmentPublicUtils class.

Two methods return an array of Attachment rows. One method returns all of the rows; the other returns only the selected rows. The rows contain all of the information that the Attachments component uses to display itself. The data is for reference purposes only, and the row returned by the method cannot be modified.

Two methods return either the total number of attachments, or the number of attachments that are currently selected.

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>oracle.apps.fnd.applcore.attachments.publicModel.AttachmentsReadOnlyRow ()</td>
<td>findSelectedAttachmentsForEvent(javax.faces.event.FacesEvent event)</td>
</tr>
<tr>
<td></td>
<td>Finds the list of rows selected in the attachment component from which the event originated. If the event is null or its source is not from a sub-component of an attachments component, or if an exception occurs during processing, an empty list is returned.</td>
</tr>
<tr>
<td>oracle.apps.fnd.applcore.attachments.publicModel.AttachmentsReadOnlyRow()</td>
<td>findAllAttachmentsForEvent(javax.faces.event.FacesEvent event)</td>
</tr>
<tr>
<td></td>
<td>Finds the list of rows selected in the attachment component from which the event originated. If the event is null or its source is not from a sub-component of an attachments component, or if an exception occurs during processing, an empty list is returned.</td>
</tr>
<tr>
<td>long</td>
<td>findAttachmentCountForEvent(javax.faces.event.FacesEvent event)</td>
</tr>
<tr>
<td></td>
<td>Finds the list of rows selected in the attachment component from which the event originated. If the event is null or its source is not from a sub-component of an attachments component, or if an exception occurs during processing, -1 is returned.</td>
</tr>
<tr>
<td>long</td>
<td>findSelectedAttachmentCountForEvent(javax.faces.event.FacesEvent event)</td>
</tr>
<tr>
<td></td>
<td>Finds the list of rows selected in the attachment component from which the event originated. If the event is null or its source is not from a sub-component of an attachments component, or if an exception occurs during processing, -1 is returned.</td>
</tr>
</tbody>
</table>

19.6.3 About Approvals

Approval functions are added to the Attachments component using the Custom Actions feature, which is documented in the previous section of this chapter.

The approvalEnabled property on the Attachments component can be used to control whether the Status column (from the FND_DOCUMENTS_TL table) is displayed in the Attachments Table. This column indicates the status of the attachment relationship between the business object and the file, not the status of the file in the content repository.
Product teams are responsible for programmatically setting the status as necessary, but are required to set this value using a lookup code provided in the `FND_ATTACHMENT_STATUSES` lookup. The following table has the full list of valid values found in this lookup table. (*Null* is also a valid value):

<table>
<thead>
<tr>
<th>Lookup Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPROVED</td>
<td>Approved</td>
</tr>
<tr>
<td>REJECTED</td>
<td>Rejected</td>
</tr>
<tr>
<td>REVIEWED</td>
<td>Reviewed</td>
</tr>
<tr>
<td>SUBMITTED_FOR_APPROVAL</td>
<td>Submitted for Approval</td>
</tr>
<tr>
<td>SUBMITTED_FOR_REVIEW</td>
<td>Submitted for Review</td>
</tr>
<tr>
<td>UNAPPROVED</td>
<td>Unapproved</td>
</tr>
</tbody>
</table>

**Note:**

The *Status* column is a read-only column when displayed in the Attachments Table.

19.6.4 About Redlining

*Redlining* is a function that shows the changes to an attachment from one version of the parent record to another. It gets its name from the format applied to the attachments that were deleted between the versions.

Redlining requires the following:

- Your business object must be tracking different versions of the same record
- There can be only one current record. All previous versions of the record are no longer editable.

At runtime, attachment views compare the attachments of the current record with that of its nominated predecessor.

- New attachments have their line prefixed with the word "Added". The *Title*, *Description*, and *Category* fields are formatted in green.
- Modified attachments have their line prefixed with the word "Edited". The *Title*, *Description*, and *Category* fields are formatted in blue. Previous values are formatted in red with a strikethrough.
- Deleted attachments have their line prefixed with the word "Removed". The *Title*, *Description*, and *Category* fields are formatted in red with a strikethrough. Deleted Attachments cannot be modified.
- Attachments that are unchanged have no prefix. The *Title* field displays in blue text.

The following figure shows an example of redlining.
19.6.4.1 How to Enable the Redlining Function

To enable redlining, do the following:

1. Fix the attachment versions for the retiring current record.

   Perform this task during the time between when the new record is created and when the current record is retired. When creating a new record, all of the attachments for the current record must be duplicated in the new record. (For information about the `duplicateAttachments` method, see About Using Attachment Utilities.) The duplication links the current and new records to the same files on the Content Server server. Once the new record becomes the current record, the attachments on the now retired record must be fixed. Use the `version` method to fix each attachment's version.

**WARNING:**

By setting the versions and then copying the attachments to the new parent record, you are copying fixed attachments and not attachments that point to the latest attachment.
Table 19-11  version Method

<table>
<thead>
<tr>
<th>Class</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void</td>
<td>To version the attachments row:</td>
</tr>
<tr>
<td></td>
<td>version(java.util.List&lt;AttachmentsVORowImpl&gt; attachments)</td>
</tr>
</tbody>
</table>

2. Configure the attachment views component to display the redlining formatting.

Set the redlineKey attribute on the attachment views component to the key of the retired record with which to compare the current record. EL Expressions are supported for retrieving the key.

As shown in the following example, the previous key is generated as a transient attribute of the parent view object. The value of the previous item key is then included as a page binding.

```xml
<fnd:attachmentView attachmentModel="#{bindings.Attachments1}" id="aview"
  redlineKey="#{bindings.PreviousItemKey.inputValue}"
  attributeCategoryVisible="true"/>
```

The supplied key must be a comma-separated concatenation of the attachment entity name and the attributes that were selected when the attachment entity was created (see Step 8 in How to Create Attachment View Links). The key can support up to five attributes. The trailing attributes do not need to be supplied if they are empty.

19.7 About Integrating Attachments Task Flows into Oracle Fusion Functional Setup Manager

Every application registers task flows with a product called Oracle Fusion Functional Setup Manager. The Functional Setup Manager provides a single, unified user interface that allows customers and implementers to configure all Oracle Fusion applications by defining custom configuration templates or tasks based on their business needs.

The Functional Setup Manager UI enables customers and implementers to select the business processes or products that they want to implement.

Function Security controls your privileges to a specific task flow, and users who do not have the required privilege cannot view the task flow. For more information about how to implement function security privileges and roles, see Implementing Function Security.

Table 19-12 lists the task flows and their parameters.

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Parameters Passed</th>
<th>Behavior</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Attachment Entities</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/attachments/publicUi/flow/ManageAttachmentEntities.xml#ManageAttachmentEntities</td>
<td>[moduleType] [moduleKey] [pageTitle]</td>
<td>Search and edit Attachment entities.</td>
<td>NA</td>
</tr>
</tbody>
</table>
Table 19-12  (Cont.) Attachments Task Flows and Parameters

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Parameters Passed</th>
<th>Behavior</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Attachment Categories</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/attachments/publicUi/flow/ManageAttachmentCategories.xml#ManageAttachmentCategories</td>
<td>[moduleType] [moduleKey] [pageTitle]</td>
<td>Search and edit Attachment categories.</td>
<td>NA</td>
</tr>
<tr>
<td>Manage Attachment File Formats</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/attachments/publicUi/flow/ManageAttachmentFileFormats.xml#ManageAttachmentFileFormats</td>
<td>None</td>
<td>Search and edit Attachment file formats</td>
<td>NA</td>
</tr>
</tbody>
</table>

For information about implementing your specific product family, do the following:

1. Access the Oracle Fusion Applications library.

2. See the Implementing Common Features guides for your product family.

19.8 About Securing Attachments

All attachments users must be assigned the "AttachmentsUser" role to use attachments. This role gives users read access to all shared file attachments. Depending on the security defined for attachments, further access privileges will be assigned to users on a per-file basis at the time of accessing the attachments.

Attachments can be secured using the Attachment Category data-security method.

A user can choose whether to Share a file when creating a file attachment. File sharing impacts the way in which files are secured. For more information, see About File Sharing.

19.8.1 About Entity Instance-Level Security

Many product teams define data security on their business objects. This data security controls which instances of that business object each user can access. Oracle Fusion attachments reuses this data security when users try to access documents directly, either via a URL or via the Content Server user interface, to ensure that users can only access files that are attached to the business object instances that they have permission to access. This security is important for controlling access to attached files when accessed directly.

19.8.1.1 About Setting Up Attachments to Uphold Entity Instance-Level Security

To enable attachments to reuse the data-security setup for your business object, you need to seed data in the DATA_OBJECT_CODE column in the FND_DOCUMENT_ENTITIES table. The value in the DATA_OBJECT_CODE column must match the value stored in the OBJ_NAME column in the FND_OBJECTS table for database resource for your business object.
Note:
Oracle Fusion attachments require product teams to have used and seeded the standard read, update, and delete functions when setting up the data security on their business objects.

For more information about Oracle Fusion Data Security, see Implementing Oracle Fusion Data Security.

19.8.2 About Attachment Category Data Security

Attachments can be secured using attachments categories. This security determines which attachments a user has access to, and what actions they can perform on that attachment, based on the Category that is assigned to it. Uptaking this security is not a mandatory requirement for product teams.

Before uptaking category data security, ensure you have assigned one or more categories to the attachment entity defined for your business object. For more information, see About Assigning Categories to the Attachment Entity.

To incorporate category data security, you need to seed data security for your attachment categories to control which roles have access to each of the categories.

The following seed data has been provided and must be used when you set up category data security:

- "FND_DOCUMENT_CATEGORIES": The seeded database resource for attachment categories data security
- Three actions (form functions) seeded for attachments. These are listed in Table 19-13.

<table>
<thead>
<tr>
<th>Action</th>
<th>FUNCTION_NAME</th>
<th>USER_FUNCTION_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seeded Read Action</td>
<td>FND_READ_APPLICATION_ATTACHMENT_DATA</td>
<td>Read Application Attachment</td>
</tr>
<tr>
<td>Seeded Update Action</td>
<td>FND_UPDATE_APPLICATION_ATTACHMENT_DATA</td>
<td>Update Application Attachment</td>
</tr>
<tr>
<td>Seeded Delete Action</td>
<td>FND_DELETE_APPLICATION_ATTACHMENT_DATA</td>
<td>Delete Application Attachment</td>
</tr>
</tbody>
</table>

19.8.2.1 How to Set Up Category Data Security

Do the following to set up category data security:

1. Define the conditions (object instance sets) that identify the category or set of categories you are securing. The CATEGORY_ID, CATEGORY_NAME, or USER_NAME condition items can be used in your condition definitions.

2. Ensure the Roles to which you want to assign data security have been created in Oracle Platform Security Services (OPSS).

3. Grant the actions that you will allow a user to perform on the set of Categories identified by your condition in Step 2. You can grant one or more of the seeded...
actions: read, update, and delete (see Table 19-13) to a role, but a role must be assigned the read action to view attachments from the Attachments UI.

The "Manage Database Resources and Policies" setup UI in the FndSetup application can be used to create these grants after you have defined your conditions in the database.

4. (Optional) The "Applications Common Reference Data Review Duty” role (GUID: 7BC1484030A9EE43499AB0EBBE17B104) provides users with read, update, and delete actions for all attachments that are assigned the "Miscellaneous" category.

To setup your own category data security for the "Miscellaneous" category, you will need to follow Steps 1 to 4.

The following condition (object instance set) is seeded with Oracle Fusion attachments and can be reused by using the "Miscellaneous” category for your product:

- **INSTANCE_SET_NAME**: FNDDOCUMENTCATEGORIESFND214
- **PREDICATE**: category_name = 'MISC'

5. Using the Manage Attachment Entities Setup UI, "switch on" category data security for your attachment entity.

For more information, see Implementing Oracle Fusion Data Security.

19.8.3 About File Sharing

File sharing impacts the way in which your files are secured. Files are stored on the Content Server as either Shared or Not Shared. Users can choose which option to use when they attach files to their business objects. For more, see About Using Attachments File-Level Security.

Shared files are files that are stored in virtual folders within the Content Server and are available to all attachments users. An attachments user is any user who has been assigned the “AttachmentsUser” role. Shared files are exempt from data security.

Unshared files are only available to those users who have access to the file via data security. Therefore, if an Oracle Fusion Applications user has privileges to access to an attachment through a business object instance and they have privileges to access attachments with a particular category, then they have access to the file in the Content Server.

Users are defined on a single Lightweight Directory Access Protocol (LDAP) server. Each of these users can log in to both Oracle Fusion Applications and Oracle Content Server using the same username and password. Users will be given appropriate privileges to each system based on their assigned roles.

19.8.4 About File Format Filtering

Most complex file formats include scripting support to assist users with automating common tasks. This has led to the rise of Cross Site Scripting (XSS attacks, where attackers gain access by including scripts in a file that an unsuspecting user then opens. The challenge, then, is how to recognize and balance the need for users to be able to include automation within their files with the need to prevent malicious scripts inflicting damage to the site.
When opening a file, the default Attachments behavior is to prompt the user to download. This removes native-browser file types from being opened in the session context. It also gives the user more control over when to open the file and, in the case of where the file is suspect, which viewer to use.

The new "Manage Attachment File Formats" page provides a site with additional options for managing the behavior of Attachments based on the file format. Since all file formats are downloaded by default, no file formats are shipped. Sites can use the management page to add file formats, identified by their mime type, either by entering their mime-type directly or by using the option to supply a sample file to source the mime-type. See About Integrating Attachments Task Flows into for more information about task flows.

File formats are set to have a state of either Deny, Download, or Open. If you do not want a file format in use on your site, you would add and set a file format to the Deny state using the management page. Subsequently, any attempt to upload a file of that type will be rejected. Any existing files of that file type will still appear in the Attachments UI, but they cannot be downloaded. Instead, an explanatory message will appear when clicking on the file link.

Setting the state to Open will open any file of a specific format when a link is clicked. Clicking on the link for these files will open based on the browser's settings.

If a file format is in a state of flux, setting the state of that file format to Download will revert it to its default behavior. This eliminates the need of having to delete the file format and re-add at a later time.

Files whose file types are not on the Open state list can still be treated individually as Open when uploaded by users who have been granted the FND_Attachment_File_Trusted_Status (ADFMethodResourceType) permission with the Invoke action. A user with this permission is able to verify that the contents of the file can be opened safely.

19.8.5 About Attachments SaaS

Software as a Service (SaaS) support has been added for Oracle Fusion attachments to ensure that the attachments belonging to organizations using the same environment are all kept completely separate to each other.

19.9 Understanding How to Use Attachments (Runtime)

This section discusses how to use Attachments file-level security, update attachments, use the Attachments update functions, and how to check file attachments out and in.

19.9.1 About Using Attachments File-Level Security

With Oracle Fusion Applications attachments, users have the ability to set attachments to be either Shared or Not Shared using the Shared option, as shown in Figure 19-28.
The Shared column is only shown in the Attachments table when the repositoryMode property is set to true. The default state for this option is Not Shared for both Text and File type attachments.

**Note:**

The option is not available for all URL and Folder type attachments, as shown in Figure 19-28. By default, all Folder type attachments are shared, and all URL type attachments are not shared. Therefore, the check box is hidden because these default values cannot be changed by the user.

All repository File or Folder attachments are shared by default. This is because only shared files are visible in the Document Picker, which is used to select the Repository Files or Folders.

If the user hovers the mouse pointer over the Shared check box, the following hint displays: Checking this box will make this file available to other users.

If the user attempts to change a File type attachment from Shared to Not Shared, the system checks to see if the file is attached to any other business object instances. If the file is attached to other business object instances, the following message is displayed and the file attachment remains shared: This change cannot be made as this file is attached to other business objects.

### 19.9.2 About Updating Attachments

Users have the ability to update attachments within the Attachments table or popup. As well as being able to update the category, title, and description of an attachment, users can update existing URL and Text attachments and replace the file of existing File attachments with a new version of the file.

**Note:**

Only one attachment can be updated at any one time.
The updateAllowed and repositoryMode properties on the Attachments component, and the Checked Out status of the Content Repository files, control how the update functionality works for a particular user.

19.9.2.1 About Attachments Update Functions

The Attachments update functions are available in the Attachments table toolbar and Actions menu.

The rendering of the update buttons in the Attachments table toolbar and the corresponding actions in the Actions menu are controlled in the following way using the updateAllowed property on the Attachments component:

updateAllowed = true:

- The Category, Title, and Description columns in the Attachments table are updateable.

  The title and description values are kept in sync with the corresponding values stored in the Content Server. When a user updates the title or description, the values are updated in the Content server when the changes are saved.

- If repositoryMode = true:
  - The Check In, Check Out, and Cancel Check Out buttons are rendered in the Attachments table toolbar, as shown in Figure 19-29.

These buttons are enabled or disabled based on the type of attachment and the checked out status of File and Text type attachments.

Figure 19-29 Attachment Table with Update Functions set to True

<table>
<thead>
<tr>
<th>Icon</th>
<th>Function</th>
<th>File</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="versioncheckout_ena.png" alt="Check Out" /></td>
<td>Check Out</td>
<td>versioncheckout_ena.png</td>
<td>repositoryMode = true</td>
</tr>
<tr>
<td><img src="versioncheckin_ena.png" alt="Check In" /></td>
<td>Check In</td>
<td>versioncheckin_ena.png</td>
<td>repositoryMode = true</td>
</tr>
<tr>
<td><img src="versionrollback_ena.png" alt="Cancel Check Out" /></td>
<td>Cancel Check Out</td>
<td>versionrollback_ena.png</td>
<td>repositoryMode = true</td>
</tr>
</tbody>
</table>
• If repositoryMode = false:
  – The Check In, Check Out, and Cancel Check Out buttons are not rendered in the Attachments table toolbar.

  **Figure 19-30  Attachment Table with Update Functions set to False**

![Attachment Table](image)

updateAllowed = false:

• None of the fields in the table are updateable when this property is set to false.

• If repositoryMode = true:
  – The Check In, Check Out, and Cancel Check Out buttons are rendered and disabled in the Attachments Table toolbar.

• If repositoryMode = false:
  – The Check In, Check Out, and Cancel Check Out buttons are not rendered in the Attachments table toolbar.

19.9.2.2 Understanding How to Determine the Checked Out Status of File and Text-Type Attachments

The Checked Out By column indicates the checked-out status in the Attachments table. This column is only rendered when the repositoryMode is set to true.

This column is always empty for URL and Folder type attachments. For File and Text attachments, this column is either:

• Empty, indicating that the file is not checked out.

• Displays the Oracle Fusion Applications username of the user who currently has the file checked out.

The user name is derived from the Checked Out By value that is stored against the file in the Content server. The stored Content Server username is mapped to the appropriate Oracle Fusion Applications username. If the value cannot be mapped, the Content server username is displayed in the column. This value is never stored in the Attachments Table.
19.9.2.3 About Enabling or Disabling Attachments Update Functions

The update functions apply to a single selected row in the Attachments table. If no rows are selected or more than one row is selected in the Attachments table, all of the update buttons and the corresponding actions in the Actions menu are disabled.

When a single row is selected in the Attachments table, the enabling or disabling of the update buttons in the Attachments table toolbar and the corresponding actions in the Actions menu are controlled using the following rules when the `updateAllowed` property is set to true:

**URL attachments**

The following rules apply regardless of the `repositoryMode` value:

- The Category, Title, and Description fields are updateable.
- The Check Out, Check In, and Cancel Check Out functions are always disabled.

**Folder attachments**

If `repositoryMode = true`:

- The Category, Title, and Description fields are updateable.
- All of the update functions are disabled.

**Text and File attachments**

If `repositoryMode = true`:

- The Category, Title, and Description fields are updateable.
- All of the update functions are disabled when the file is checked out by a user other than the current user.
- The Check Out function is disabled, and the Check In, and Cancel Check Out functions are enabled when the file is checked out by the current user.
- The Check Out function is enabled, and the Check In and Cancel Check Out functions are disabled when the file is not checked out.

19.9.3 How to Check Out and Check In File Attachments

This section describes how to check out and check in file attachments.

To check out and check in file attachments, do the following:

1. Highlight the File attachment in the Attachments table. Click the Check Out button located on the toolbar.

   The file is immediately checked out in the Content Repository. The Checked Out By column is updated to show your user name.

   **Tip:**
   
   Clicking the Cancel Check Out button cancels the check out without making any updates to the file. The check out is cancelled immediately in the Content Repository.

2. Go to your local file system and make the necessary updates to this file.
3. Return to the Attachments table and highlight the checked out file. Click the **Check In** button located on the toolbar to open the Check In File dialog, as shown in Figure 19-31.

![Figure 19-31 Check In File Dialog](image)

4. Click **Browse** to browse your local file system. Select the updated file that you want to upload.

5. Click **OK** to save your changes and close the dialog.

   The selected file is uploaded to the content server. The **Checked Out By** column is cleared.

6. Click **Save** on the Attachments Table page to commit your transaction.

   **Tip:**

   If you cancel the entire transaction, the checked out status returns to the state that it was in before the transaction took place.
This chapter describes how to create, edit, and delete tree structures, trees, and tree versions, and how to develop applications using trees.

The chapter includes the following sections:

- Introduction to Trees
- About Configuring the Trees Application Launch Page
- Working with Tree Structures
- Working with Trees
- Working with Tree Versions
- Managing Labels in the Generic Label Data Source
- About Using the Applications Hierarchy Component to Develop Applications
- About Integrating Custom Task Flows into the Applications Hierarchy Component
- About Using the fnd:hierarchy Property Inspector to Specify Tree Versions
- About Using the Expression Builder to Bind TreeCode, TreeStructureCode, and TreeVersionId Properties
- Embedding the Tree Picker Component in a User Interface
- About Setting Bind Variables and View Criteria
- About Using Service APIs to Manage Trees
- Advanced Topics

20.1 Introduction to Trees

Oracle Fusion tree management allows data in applications to be organized into a hierarchical fashion, and allows you to create tree hierarchies based on specific data.

Here are some of the advantages of how using tree hierarchies to develop applications can help you:

- *Reusable code* that results in a one-time-only implementation of many tree-management features, and can be used immediately by every type of application hierarchy.
• Open metadata that can be read by any application that needs to use tree-management hierarchies. This does the following:
  – Minimizes the number of application programming interfaces (APIs) that need to be written for accessing hierarchies
  – Allows the sharing and understanding of hierarchies across multiple applications
  – Allows the sharing of hierarchies with Oracle Business Intelligence reporting and analytics systems

• Tree structures that capture the business rules the data must adhere to.

• ADF Business Components view objects that are used as data sources, eliminating the need to build new types of data sources.

• Hierarchical relationships between entities that are external to the entity itself, allowing multiple hierarchical views to be implemented for the same set of entities. Each of these hierarchies can be used to implement a different business function.

• Data flattening that improves query performance against the hierarchical data, especially for hierarchical queries such as roll-up queries.

• Business events that can be consumed by any application requiring additional processing on specific tree operations.

• Tree and node-level access control that eliminates the need for product teams to write their own access-control code.

• Well-defined APIs available for metadata and data that make it easy for the Oracle Fusion Upgrade Office to write migration tools for existing hierarchies in E-Business Suite, PeopleSoft, and Siebel.

As a developer, you will work mostly with tree structures. The task of working with trees and tree versions normally will fall to customers. However, since you probably also will be required to work with trees and tree versions, both types of tasks are described in this chapter.

20.1.1 Understanding Tree Structures, Trees, and Tree Versions

A tree structure is a way of describing a hierarchy. A tree is an instance of this hierarchy. Every tree structure contains a tree. Trees may have one or more versions. Each tree version contains at least one root node; that is, a member that has no superior. (Occasionally, a tree version may have more than one root node.) The lines connecting elements in a tree structure are branches; the elements themselves are nodes.

The names of relationships are modeled after family relations:

• A node is a parent of another node if it is one step higher in the hierarchy and closer to the root node.

• Sibling nodes share the same parent node.

For example, in the following picture, XYZ Corp. is the parent of Marketing and Finance, which are its children. Accounts Receivable and Accounts Payable are siblings, and are the children of Finance.
In Oracle Fusion tree management, a tree structure defines a group of common business rules for a family of trees, for example, Department, Account, or Project, and allows an application to select and enable a subset of trees to fulfill a specific purpose in that application.

A tree contains data that is organized in a hierarchy, allowing the creation of groupings and rollups of information that already exist within an organization. A tree can have one or more tree versions. Typically, when changes are made to an existing tree, a new version is created and published.

A tree structure data source supplies the data for a tree through its nodes. Multiple data sources can be associated with a tree structure and can have well-defined relationships among them. Using the example in the previous picture, the Accounts Receivable data source is a child of the Finance data source. Data sources also support business rules that define how the data from a data source participates in a tree.

The following table lists other commonly used tree terms and their descriptions.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth</td>
<td>The depth of a node is the length of the path from the root to the node. The root node is at depth zero.</td>
</tr>
</tbody>
</table>
| Label          | Allows for a storage of "tags" that can be used on each tree node in a tree. There are three labeling schemes:  
                 | • Level - Labels that are automatically assigned based on the data source that the tree node belongs to. A level label points to a specific data source.  
                 | • Group - Labels that a user can assign to tree nodes arbitrarily.  
                 | • Depth - Labels that are automatically assigned based on the depth of the tree node within the tree. No manual assignment is performed. Note that in an unbalanced hierarchy, a level may not be equal to depth.  
                 | Labels can be stored in any table and the label data source is registered with the tree structure. |
| Tree label     | When a labeling scheme is used for trees, the selected labels are stored in the tree label entity and each tree node references a tree label. See "Label." |
| Node           | A logical term that refers to the actual data, whatever that may be. Technically, the node may be stored either in a product-specific table or in an entity that has been established by the Tree Management solution as the default storage mechanism. However, since all data in Oracle Applications usually already has a storage home, only customers should store the node in an entity. |
| Tree node      | A node that is included in a tree. |

Figure 20-1 Example of a Tree

XYZ Corp.

Marketing

Finance

Accounts Receivable

Accounts Payable
Table 20-1  (Cont.) Common Tree Terminology

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree node type</td>
<td>A tree node has a node type. Node types can be any one of the following:</td>
</tr>
<tr>
<td></td>
<td>• Single - Indicates that the node is a value by itself. For example, a tree node for Employee “Larry</td>
</tr>
<tr>
<td></td>
<td>Ellison” or Employee “Steve Jobs” in an employee hierarchy.</td>
</tr>
<tr>
<td></td>
<td>• Range - Indicates that the node represents a range of values and possibly could have many children.</td>
</tr>
<tr>
<td></td>
<td>For example, a tree node representing account numbers 10000 to 99999.</td>
</tr>
<tr>
<td></td>
<td>• Referenced Tree - Indicates that the tree node is actually another tree whose nodes are not physically</td>
</tr>
<tr>
<td></td>
<td>stored in this tree. For example, a geographic hierarchy for the United States can be referenced in</td>
</tr>
<tr>
<td></td>
<td>a World geographic hierarchy.</td>
</tr>
<tr>
<td>Tree levels</td>
<td>Provide a way to organize tree nodes. In most trees, all nodes at the same level represent the same</td>
</tr>
<tr>
<td></td>
<td>kind of information. For example, in a tree that reflects the organizational hierarchy, all division</td>
</tr>
<tr>
<td></td>
<td>nodes appear on one level and all department nodes on another. Similarly, in a tree that organizes a</td>
</tr>
<tr>
<td></td>
<td>user’s product catalog, the nodes representing individual products might appear on one level and the</td>
</tr>
<tr>
<td></td>
<td>nodes representing product lines on the next higher level.</td>
</tr>
<tr>
<td></td>
<td>When levels are not used, the nodes in the tree have no real hierarchy or reporting structure but do</td>
</tr>
<tr>
<td></td>
<td>form a logical summarization structure. Strictly enforced levels mean that the named levels describe</td>
</tr>
<tr>
<td></td>
<td>each node’s position in the tree. This is natural for most hierarchies.</td>
</tr>
<tr>
<td></td>
<td>Loosely enforced levels mean that the nodes at the same visual level of indentation do not all</td>
</tr>
<tr>
<td></td>
<td>represent the same kind of information, or nodes representing the same kind of information appear at</td>
</tr>
<tr>
<td></td>
<td>multiple levels. With loosely enforced levels, users assign a level to each node individually; the</td>
</tr>
<tr>
<td></td>
<td>level is not tied to a particular visual position.</td>
</tr>
<tr>
<td>Tree structure access</td>
<td>The set of rules that control access to a tree structure.</td>
</tr>
<tr>
<td>Tree access</td>
<td>The set of rules that control access to a tree.</td>
</tr>
<tr>
<td>Tree node access</td>
<td>The set of rules that control access to a particular node (and its subtree) within a given tree version.</td>
</tr>
<tr>
<td>Effective dates</td>
<td>Enable users to specify new objects, departments, reporting relationships, or organizational structures</td>
</tr>
<tr>
<td></td>
<td>in advance and have them take effect automatically. Users also can use trees with past, present, or</td>
</tr>
<tr>
<td></td>
<td>future effective dates when reporting on current or historic data.</td>
</tr>
<tr>
<td>Reference data set</td>
<td>A value that determines which reference data set will be used for each reference data object.</td>
</tr>
<tr>
<td>determinant (external)</td>
<td>Business units, regulatory regions, and reference data sets all can determine which reference data sets</td>
</tr>
<tr>
<td></td>
<td>are valid for the creation of a transaction or reference data object.</td>
</tr>
<tr>
<td>Audit</td>
<td>A process that runs a series of tests against tree metadata and tree data to validate its integrity.</td>
</tr>
</tbody>
</table>

20.2 About Configuring the Trees Application Launch Page

Before you can manage tree structures, trees, and tree versions using the web-browser-based trees application, you must create the application launch page in Oracle JDeveloper. The launch page contains links to the Tree Structures, Trees and Tree Versions, and Manage Labels applications, which contain the management task flows you will use.
You also will need to perform additional steps that are required to schedule the concurrent processes that the trees application uses for audit and flattening.

**Before you begin:**
Create an application initialized for use with Oracle Middleware Extensions for Applications. For more information, see Chapter 2, "Setting Up Your Development Environment."

### 20.2.1 How to Create the Launch Page

To create the launch page, do the following:

1. Configure a UIShell launcher page for your ViewController project using the procedure described in Populating a UI Shell.

2. Add a taskflow entry in the ADF menu as a node with the following properties:
   - **focusViewId** - /jspx file
   - **id** - tree_<jspx file>
   - **Label** - Trees and Tree Versions
   - **Task Type** - dynamicMain
   - **taskFlowId** - /WEB-INF/oracle/apps/fnd/applcore/trees/ui/taskflow/TreeStructureSummary.xml#TreeStructureSummary

3. Repeat Steps 1 and 2 to create a second itemNode with the following properties:
   - **focusViewId** - /jspx file
   - **id** - tree_<jspx file>
   - **label** - Trees and Tree Versions
   - **Task Type** - dynamicMain
   - **taskFlowId** - /WEB-INF/oracle/apps/fnd/applcore/trees/ui/taskflow/TreeSummary.xml#TreeSummary

4. Repeat Steps 1 and 2 to create a third itemNode with the following properties:
   - **focusViewId** - /jspx file
   - **id** - tree_<jspx file>
   - **label** - Manage Labels
   - **Task Type** - dynamicMain
   - **taskFlowId** - /WEB-INF/oracle/apps/fnd/applcore/trees/ui/taskflow/FndLabelSummary.xml#FndLabelSummary

5. Ensure you have an itemNode for defaultRegional. If you do not, define one with the following properties:
   - **focusViewId** - /jspx file
   - **id** - __<jspx file>_itemNode__FndTasksList
Note:
Use double underscores where indicated.

- **label** - #{applcoreBundle.TASKS}
- **Task Type** - defaultRegional
- **taskFlowId** - /WEB-INF/oracle/apps/fnd/applcore/patterns/uishell/ui/publicFlow/TasksList.xml#TasksList
- **Disclosed** - true

6. Click **OK** in the Set Run Configuration window.

The trees application launch page opens in a browser window, as shown in the following picture.

**Figure 20-2   Trees Application Launch Page**

7. Do one of the following:

   - Click the **Tree Structures** link to open the Tree Structure summary page.
   - Click the **Trees and Tree Versions** link to open the Trees summary page.
   - Click the **Central Labels** link to open the Central Labels summary page.
   - Click the **Trees Picker** link to open the Trees Picker application.

The following picture shows the launch page with all applications open, and the Tree Structure application's summary page displayed.
20.3 Working with Tree Structures

Tree Structures define hierarchical relationships between information stored in one or more data sources. Working with tree structures includes the following tasks:

- Managing tree structure data sources
- Specifying data source parameters
- Searching for, creating, duplicating, editing, or deleting tree structures
- Setting tree structure status
- Auditing tree structures.

A flag named \texttt{PROTECTED\_FLAG} is available for the tree structures. By default, this flag is set to \texttt{N}. If you set this flag to \texttt{Y}, you cannot modify the tree structure. When you want to protect a tree structure from being updated, you can set this flag to Yes. A protected tree structure can be modified or deleted only by the \texttt{SEED\_DATA\_FROM\_APPLICATION} user.

You can create an exclusion list and tree structures to that list. Any tree structure that is a part of the exclusion list will not be considered for export. The tree structures which have been excluded can be populated by writing an SQL statement.

20.3.1 About Managing Tree Structure Data Sources

Tree structure data sources provide the data items for a hierarchy. In the tree-management infrastructure, these are ADF Business Components view objects. You should define view objects for all the intended data sources before setting up a tree structure. For each view object attribute that is to be displayed in the hierarchy column
of an ADF Faces Tree or ADF Faces TreeTable, the Application property 
HierarchyDisplay is set to true.

Figure 20-4 View Object Attributes: HierarchyDisplay Property

Tree management provides a generic data source for holding nodes: 
oracle.apps.fnd.applcore.trees.model.view.FndNodeVO. This data source 
may be used for tree-only nodes, that is, nodes that do not exist in any other entity in 
the system. Likewise, a generic label data source has also been provided: 
oracle.apps.fnd.applcore.trees.model.view.FndLabelVO.

20.3.2 About Specifying Data Source Parameters

Tree data sources have optional data source parameters with defined view criteria and 
associated bind variables. You can specify view criteria as a data source parameter 
when creating a tree structure, and edit the parameters when creating a tree.

Note:

Parameter values customized at the tree level will override the default values 
specified at the tree-structure level.

The parameters will be applied when performing node operations, and the display of 
the nodes in the hierarchy, for any tree version under that data source. Data source 
parameters also provide an additional level of filtering for different tree structures.

Tree management supports three data source parameter types:

- VIEW_CRITERIA - Used to capture the view criteria name, which will be applied 
to the data source view object
- BOUND_VALUE - Used to capture any bound value, which will be used as part 
of the view criteria condition
- VARIABLE - Used to capture and bind variable that is being used by the data 
source view object, particularly for WHERE clause support

In addition to parameter values provided by the customer, tree management provides 
support for those special parameters whose values for any bind variable are seeded at 
runtime by tree management.

For example, to use the effectiveStartDate attribute of a tree version that a data source 
uses as one of the bind variables from which the value for the effectiveStartDate bind
variable will be retrieved from the trees effective start date, you can specify a data source parameter `effectiveStartDate` with the value 
`#{treeVersion.effectiveStartDate}`. You would then need to expose an `effectiveStartDate` bind variable for the data source view object either in view criteria or a WHERE clause.

You can specify parameters using the syntax for value and name shown in the following table.

**Table 20-2 Parameter Syntax**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree Structure</td>
<td><code>#{treeStructure.ATTR_NAME_WITH_FIRSTCHAR_IN_LOWER_CASE}</code> For example, <code>#{treeStructure.treeStructureCode}</code>.</td>
</tr>
<tr>
<td>Tree</td>
<td><code>#{tree.ATTR_NAME_WITH_FIRSTCHAR_IN_LOWER_CASE}</code> For example, <code>#{tree.treeCode}</code>.</td>
</tr>
<tr>
<td>Tree Version</td>
<td><code>#{treeVersion.ATTR_NAME_WITH_FIRSTCHAR_IN_LOWER_CASE}</code> For example, <code>#{treeVersion.treeVersionId}</code>.</td>
</tr>
</tbody>
</table>

**Note:**

Binding parameters are supported for String, Number, and Date data types only.

In 11gR1, tree management does not support View Criteria and Variable when used in combination, or multiple View Criteria as data source parameters.

### 20.3.2.1 About Implementing Use Cases

This section includes an example use case and discusses basic use cases and their settings.

**20.3.2.1.1 Example Use Case**

The data source DemoEmpVO has the view criteria DemoEmpVC1, which is based on the bound values DemoEmpBV1 and DemoEmpBV2. These are to be applied to the data source view object for tree versions under the DEMO_EMP_TS tree structure, with varying bound values DemoEmpBV1 and DemoEmpBV2 for trees under this tree structure.

When creating the tree structure FND_DEMO_EMP_TS, the following parameters must be added to the tree structure data source that corresponds to DemoEmpVO:

- Parameter name: `VIEW_CRITERIA_NAME`
- Parameter type: View Criteria
- Parameter value: `<name of the view criteria to be applied>`, in this case DemoEmpVC1

The following two bindings also must be added:

- Parameter names: DemoEmpBV1 and DemoEmpBV2
- Parameter type: Bound Value
Parameter value: `<actual value, which can be overridden at tree level>`

Note:
Binding parameters are supported for String and Number data types only.

20.3.2.1.2 Basic Use Cases and Their Settings
The following are examples of use cases and settings that you can implement using the parameter infrastructure for tree structure data sources.

Data source having a view criteria defined with a bind variable:

*Figure 20-5  View Object Setup Wizard*
Data source has a WHERE clause using a bind variable:

![Figure 20-6  Parameters in Data Source Parameter UI](image1)

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Parameter Type</th>
<th>Parameter Value</th>
<th>Mandatory</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeptNoBV</td>
<td>Bind Variable</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>VIEW_Criteria_NAME</td>
<td>View Criteria</td>
<td>ParamVOEmpWO</td>
<td></td>
</tr>
</tbody>
</table>

![Figure 20-7  View Object Setup Wizard](image2)
Figure 20-8 Parameters in Data Source Parameter UI

Data source has a view criteria defined with a bind variable for special parameters:

Figure 20-9 View Object Setup Wizard (View Criteria)
Data source has a WHERE clause using a bind variable for special parameters:

**Figure 20-11  View Object Setup Wizard (WHERE Clause)**
20.3.3 How to Search for a Tree Structure

If you wish to duplicate, edit, or delete an existing tree structure and it is not currently visible in the results list, you can search for it using the following procedure. The procedure assumes that the Tree Structure summary page is open in your web browser.

To search for an existing tree structure, do the following:

1. In the Search area of the page, construct a search using any or all of the following search criteria:
   - Code
   - Name
   - Status

2. Click **Search**.

   All tree structures matching your search criteria appear in the Results area of the page.

   Click **Advanced** to perform an advanced search by specifying additional options for search. You also can save your search criteria for future use.

20.3.4 About Using the Search Field

Throughout the trees application you will see a search field located to the right of many field names, as shown in the following picture.

![Search Field](Image)

Click the down arrow to display a dropdown list that contains the available values for that field. You can select from the list, or search for other values. For example, the following picture shows the dropdown list that displays when you click the down arrow associated with the **Application** search field found on the Create Tree Structure: Specify Definition page.
From each search field dropdown list, you can do one of the following:

- Select a value from the list that displays.
- Click the **Search** link to search for a value that does not exist in the list.

If you select a value from the list, the dropdown list closes and that value appears in the search field.

If you click the **Search** link, a search-and-select window similar to the one shown in the following picture opens:
You now can search for a value and then click OK to select it.

20.3.5 How to Create a Tree Structure

The following procedure explains how to create a new tree structure. The procedure assumes that the Tree Structure summary page is open in your web browser.

To create a tree structure, do the following:

1. Click the Create icon, or choose Create from the Actions dropdown menu.

   The Create Tree Structure: Specify Definition page, shown in the following picture, opens.

   ![Create Tree Structure: Specify Definition Page (1)]

2. Enter a code for the tree structure.

   The code can be any combination of alphanumeric characters, but cannot contain more than 30 characters. Codes are used in APIs to work with trees, and uniquely identify the tree structure metadata. You cannot create a tree structure with ORA_ as prefix in code. ORA_ is a reserved word.

3. Enter a name for the tree structure.

   The name is a user-friendly name for a tree structure. It appears only in graphical user interfaces (GUIs), and cannot contain more than 80 characters.

4. Enter the name of an appropriate application, or click the down arrow to select or search for one.

5. Enter a description.

6. Enter the name of an appropriate tree node table, or click the down arrow to select or search for one.
If you enter the name of a custom tree node table or select a tree node table other than the FND_TREE_NODE default, the page re-displays with a new (optional) field asking you to enter a view object definition name for that custom tree node table. This field is shown in the following picture.

**Figure 20-17  Create Tree Structure: Specify Definition Page (2)**

7. Select a tree-sharing method.
   - **Open** - indicates that the tree will be associated with all Set IDs
   - **Set ID** - indicates that the tree will be associated with a specific Set ID.

8. Select a creation mode:
   - **Customer** - indicates that the customer is creating the tree structure
   - **Oracle** - indicates that an Oracle developer is creating the tree structure.

9. Select **Customizable** if the tree structure can be customized by the customer.

10. Select **Allow Multiple Active Tree Versions** to allow two or more tree versions to be in an ACTIVE state for the same date range.

11. Select a versioning type for editing tree versions to specify whether the user is allowed to edit an active tree version or must create a new one.

12. Click **Next**.

   The Create Tree Structure: Data Sources page, shown in the following picture, opens.
13. Select a labeling scheme:

- **None** - Specifies that no labeling scheme will be used.
- **Level based** - Specifies a label that points to a specific data source.
- **Group based** - Specifies a label that the user can assign arbitrarily.
- **Depth based** - Specifies a label that the depth-from-top parent decides to display. No manual assignment is performed.

If you choose a level-based, depth-based, or group-based labeling scheme, the Labeling Scheme area of the page changes, displaying additional options, as shown in the following picture.

**Figure 20-19  Additional Labeling Scheme Options**

14. Select any or all of the following:

- **Date Range** - Specifies whether a selection of nodes should be restricted to the same date range as the tree version.
- **Allow Multiple Root Nodes** - Allows you to add multiple root nodes when creating a tree version.
- **Set ID** - Specifies whether a selection of nodes should be restricted to the same set as the tree. (All versions belong to the same set.)
- **Allow Ragged Nodes** (for Labeling Schemes Level based, Depth based, or Group based only) - Specifies whether a hierarchy can be unbalanced; that is, if
it can contain nodes that are not leaf nodes and contain no children. In the
following picture, "ICs" does not have any children, making its rooted path
shorter than all the others in the hierarchy.

**Figure 20-20  Example of a Ragged Hierarchy**

- **Allow Skip-Level Nodes** (for Labeling Schemes Level based, Depth based, or
  Group based only) - Specifies whether a hierarchy can have two nodes at the
  same level with parent nodes at different levels. In Figure 20-21, Washington,
  DC, does not have a node at the State level.

**Figure 20-21  Example of a Skip-Level Hierarchy**

- **Optional Label Enforcement** - Specifies whether label is enforced in the audit
  of the tree structure. By default, this option is not selected and specifying a
  label data source is enforced while adding a data source for the tree structure.
  Alternatively, if this option is selected, **Label Data Source** in the Add Data
  Source page is not an optional field.

15. Click the **Add** icon.

The Add Data Source window, shown in the following picture opens.

**Figure 20-22  Add Data Source Window**
If you chose a level-based, depth-based, or group-based labeling scheme, the top portion of the page changes, displaying an additional Label Data Source field, as shown in the following picture. The Label Data Source will be optional if Optional Label Enforcement option is selected, else this is a mandatory field.

Figure 20-23 Additional Add Data Source Field

16. Enter the name of the view object.

17. Optionally, enter a name.

18. If you do not enter a name, the same name as that of the view object is copied to this field and is used.

19. Select a maximum depth value from the dropdown list.

   Maximum depth specifies how many levels are allowed. For example, in Project[max depth=2] > Task[max depth=infinite], one project, one sub-project, and an infinite number of tasks are allowed.

20. Enter the name of the label data source (for Level based, Depth based, or Group based labeling schemes only).

   The view object oracle.apps.fnd.applcore.trees.model.view.FndLabelVO can be used as an ad-hoc label data source. For more information about label data sources, see Managing Labels in the Generic Label Data Source.

21. Select any or all of the following:

   - Use non defined primary key columns - indicates that you can specify other attributes as primary key columns. If not selected, existing primary keys defined as primary key columns for a data source view object will be used.

     If selected, the additional fields shown in the following picture display.

     Figure 20-24 Primary Key Columns

     - Allow use as leaves - indicates that data from the data source can form a leaf

     - Allow duplicates - indicates that a data item can exist multiple times in the same hierarchy. For example, in an "Item" hierarchy for an automobile, a particular bolt may appear multiple times in the hierarchy.

     - Use as free node - indicates that this node can be used as a free node.
• Select a Usage Limit:

  – **None** - Specifies that there are no restrictions.
  
  – **Use all values** - Specifies that all available nodes must be included in the tree version.
  
  – Select a Child Definition:

• **Allow range children** - indicates that the hierarchy supports nodes that are a range of values.

• **Allow linked foreign key children** - indicates that the relationship is external to Tree Management.

If you select this option, you also can select a **View Link Accessor** value from the dropdown list that displays, as shown in the following picture.

*Figure 20-25  View Link Accessor Dropdown List*

22. Click **Add** under **Data Source Parameters**.

The window now displays data source parameters text-entry fields, as shown in the following picture.

*Figure 20-26  Data Source Parameters Text-Entry Fields*

23. Enter a parameter, select a parameter type, and enter a value.
When specified, a parameter applies to every version under that tree. Parameter values also can be overridden at the tree level. For more information, see About Specifying Data Source Parameters.

24. Select Mandatory if the parameter is to be required.

25. Click OK.

The Create Tree Structure: Specify Data Sources page refreshes, displaying the view object, as shown in the following picture.

*Figure 20-27  Create Tree Structure: Specify Data Sources Page with View Object*

26. Click Next.

The Create Tree Structure: Specify Performance Options page opens, as shown in the following picture.
27. Enter the name of an appropriate row-flattened table or click the down arrow to select or search for one.

28. Enter the name of an appropriate column-flattened table or click the down arrow to select or search for one.

29. Enter the name of an appropriate column-flattened entity object if the field does not already contain one.

30. Enter the name of an appropriate BI View Object.

31. Click Next.

   The Create Tree Structure: Specify Access Rules page opens.

   **Note:**

   The Create Tree Structure: Specify Access Rules page is not yet implemented.

32. Click Submit.

33. Click OK to close the Confirmation window.

   The Create Tree Structure Confirmation Window opens, as shown in the following picture.

   **Figure 20-29  Create Tree Structure Confirmation Window**

20.3.6 How to Duplicate a Tree Structure

Duplicating a tree structure simply copies the metadata definition from an existing tree structure to the duplicate. This operation does not copy the underlying tree and tree versions defined for the source tree structure.
To duplicate a tree structure, do the following:

1. Select the tree structure you want to duplicate.

   See How to Search for a Tree Structure, if the tree structure you want to duplicate is not in the current Results list.

2. Click the Duplicate icon, or choose Duplicate from the Actions dropdown menu.

   The Create Tree Structure window opens, as shown in the following picture:

   ![Create Tree Structure Window](image)

3. Enter a new name for the duplicate tree structure if you want to replace the name that already displays in the field.

4. Enter a duplicate tree structure code if you want to replace the code that already displays in the field.

5. Click Save and Close to create the duplicate.

### 20.3.7 How to Edit a Tree Structure

When you edit an existing tree structure, you simply step through many of the same pages you used to create a tree structure.

To edit an existing tree structure, do the following:

1. Select the tree structure you want to edit.

   See How to Search for a Tree Structure, if the tree structure you want to duplicate is not in the current Results list.

2. Do one of the following:

   - Click the Edit icon.
   - Choose Edit from the Actions dropdown menu.
   - Click the tree-structure name.

3. Edit the appropriate data on the Edit Tree Structure: Specify Definition page.

4. Click Next.

5. Do any of the following:

   - Edit the data on the Edit Tree Structure: Specify Data Sources page.
   - Click Add to add another view object.
• Select an existing view object and click **Edit** to edit it.
• Select an existing view object and click the Delete icon to delete it.

6. Click **Next**.

7. Edit the appropriate data on the Edit Tree Structure: Specify Performance Options page.

8. Click **Next**.

   The Edit Tree Structure: Specify Access Rules page opens.

   **Note:**
   The Edit Tree Structure: Specify Access Rules page is not yet implemented.

9. Click **Submit**.

10. Click **OK** to close the Confirmation window.

**20.3.8 How to Delete a Tree Structure**

Deleting a tree structure also deletes all associated tree and tree versions defined under that specific tree structure.

To delete a tree structure, do the following:

1. Select the tree structure you want to delete.

   See **How to Search for a Tree Structure**, if the tree structure you want to delete is not in the current Results list.

2. Click **Delete**, or choose **Delete** from the **Actions** dropdown menu.

   The Delete Tree Structure warning window opens, as shown in the following picture:

   **Figure 20-31  Delete Tree Structure Warning Window**

3. Do one of the following:

   • Click **No** to cancel the operation.
   • Click **Yes** to delete the tree structure.

**20.3.9 How to Set Tree Structure Status**

Changing the status of a tree structure also changes the status of the trees and tree versions contained in that tree structure. You can set the status of a tree structure to any one of the following:
• Draft
• Active
• Inactive

Setting a tree structure's status to Active automatically triggers an audit of that tree structure. See the following section for more information about auditing.

To set the status of a tree structure, do the following:

1. Select a tree structure.
   See How to Search for a Tree Structure, if the tree structure is not in the current Results list.

2. Choose the appropriate status option from the Actions > Set Status dropdown menu.

3. Click OK to close the Warning window. The Warning window appears only when you want to set the status as Draft or Inactive.

4. Click OK to close the Confirmation window.

20.3.10 About Auditing a Tree Structure

Auditing tree-structure metadata verifies that it conforms to all rules and ensures data integrity. Running an audit allows you to view audit details and messages, and to correct any validation errors that the audit detects.

Setting a tree structure's status to Active automatically triggers an audit of that tree structure. You also can audit a tree structure manually.

The following table describes what each validator checks for, as well as possible reasons why each validator might fail.

<table>
<thead>
<tr>
<th>Table 20-3 Validator Descriptions</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Validator</th>
<th>Checks for...</th>
<th>Validation may have failed because...</th>
<th>To correct...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restrict by SetID Validator</td>
<td>If the tree structure has Restrict Tree Node List of Values Based on SetID flag set to Yes, each of its data source view objects must have a SetID attribute. This restriction does not apply when the flag is set to No.</td>
<td>The tree structure has Restrict Tree Node List of Values Based on SetID flag = Y, but one or more of its data source view objects do not contain a SetID attribute.</td>
<td>Consult the owning developer. If SetID restriction is desired for this tree structure, ensure your developer has included a SetID attribute on all data sources. If SetID restriction is not desired, ensure your developer sets the flag to No.</td>
</tr>
<tr>
<td>Validator</td>
<td>Checks for...</td>
<td>Validation may have failed because...</td>
<td>To correct...</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| Row Flattened Table Name Validator | A valid "Row Flattened Table" should be specified for the tree structure on the "Specify Performance Options" page. It can be the standard row flattened table FND_TREE_NODE_RF, or a custom table can be specified.                                                                 | • No table is specified in "Row Flattened Table" on the "Specify Performance Options" page of the Manage Tree Structures UI.  
• The specified table does not exist in the database.  
• The specified table does not contain the same columns that FND_TREE_NODE_RF table contains.                                                                                             | Consult the owning developer to correct the row flattened table definition.                                                                                                                                                                                      |
| Available Label Data Sources Validator | If the tree structure has a Labeling Scheme specified, the label data source view object specified for each data source must be accessible and the primary keys must be valid.  
This restriction does not apply when the Labeling Scheme has been set to None.                                                                                                               | • Any of the specified label data source view objects do not exist.  
• Any of the specified label data source view objects do not have primary keys.  
• At the time a label data source view object is initially defined, the backend registers the primary keys for the view object at that time. If the view object is later modified such that its primary keys no longer match the primary keys that were registered earlier, this validator will fail.  
• Consult the owning developer to correct the label data source view object specified.  
• Consult the owning developer to correct the primary keys of the label data source view object specified.  
• Consult the owning developer to either correct the primary keys in the label data source view object to match the primary keys that were earlier registered in FND_TS_DATA_SOURCE, or correct the primary keys registered in that table to match the new view object definition. |
### Table 20-3 (Cont.) Validator Descriptions

<table>
<thead>
<tr>
<th>Validator</th>
<th>Checks for...</th>
<th>Validation may have failed because...</th>
<th>To correct...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available Data Sources Validator</td>
<td>Each data source view object specified for the tree structure must be accessible and all its primary key attributes should be valid.</td>
<td>• Any of the specified data source view objects do not exist.</td>
<td>• Consult the owning developer to correct the data source view object specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• At the time a data source view object is initially defined, the backend registers the primary keys for the view object at that time automatically if “Use non-defined primary key columns” is not selected. If it is selected, the backend registers the primary keys specified explicitly by the user on the Add Data Source page. The validator will fail if the registered primary keys contain any duplicates.</td>
<td>• Consult the owning developer to correct the duplicate column in the registered primary keys.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• “Use non defined primary key columns” is checked in a data source, but the list of specified primary key columns does not match the primary keys defined in the corresponding data source view object.</td>
<td>• Consult the owning developer to correct the primary keys of the data source view object specified.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Any common attributes that exist in both the data source view object and the tree node view object are not of the same data type in both view objects.</td>
<td>• Consult the owning developer to correct any mismatch in data types.</td>
</tr>
<tr>
<td>Column Flattened Table Name Validator</td>
<td>A valid &quot;Column Flattened Table&quot; should be specified for the tree structure on the &quot;Specify Performance Options&quot; page. It can be the standard row flattened table FND_TREE_NODE_CF, or a custom table can be specified.</td>
<td>• No table is specified in &quot;Column Flattened Table&quot; field on the &quot;Specify Performance Options&quot; page of the Manage Tree Structures UI.</td>
<td>Consult the owning developer to correct the column flattened table definition.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The specified table does not exist in the database.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The specified table does not contain the same columns that FND_TREE_NODE_CF table contains.</td>
<td></td>
</tr>
</tbody>
</table>
Table 20-3 (Cont.) Validator Descriptions

<table>
<thead>
<tr>
<th>Validator</th>
<th>Checks for...</th>
<th>Validation may have failed because...</th>
<th>To correct...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Restrict by Date Validator</strong></td>
<td>If the tree structure has Restrict Tree Node List of Values Based on Date flag set to Yes, each of its data source view objects must have Effective Start Date and Effective End Date attributes.</td>
<td>The tree structure has Restrict Tree Node List of Values Based on Date flag = Y, but one or more of its data source view objects do not contain EffectiveStartDate and EffectiveEndDate attributes.</td>
<td>Consult the owning developer. If the date restriction is desired for this tree structure, ensure your developer has included an EffectiveStartDate and EffectiveEndDate attribute on all data sources. If the date restriction is not desired, ensure your developer sets the flag to No.</td>
</tr>
<tr>
<td><strong>Tree Node Table Name Validator</strong></td>
<td>A valid &quot;Tree Node Table&quot; should be specified for the tree structure on the 'Specify Performance Options' page. It can be the standard row flattened table FND_TREE_NODE, or a custom table can be specified.</td>
<td>No table is specified in &quot;Tree Node Table&quot; field when editing a Tree Structure. The specified table does not exist in the database. The specified table does not contain the same columns that the standard FND_TREE_NODE table contains.</td>
<td>Consult the owning developer to correct the tree node table definition.</td>
</tr>
<tr>
<td><strong>Allow Node Level Security Validator</strong></td>
<td>If &quot;Allow Node Level Security&quot; flag is set to N for the tree structure, the same flag cannot be set to Y on any of its data sources. This is a backend setting that is not viewable through the Manage Tree Structures page.</td>
<td>&quot;Allow Node Level Security&quot; flag is set as No for the tree structure, but one or more associated data sources have that flag set to Yes.</td>
<td>Consult the owning developer to correct the &quot;Allow Node Level Security&quot; flags in the tree structure and/or its data sources.</td>
</tr>
</tbody>
</table>

20.3.10.1 How to Audit a Tree Structure Manually

To edit a tree structure manually, do the following:

1. Select a tree structure.

   See How to Search for a Tree Structure, if the tree structure is not in the current Results list.

2. Choose Audit from the Actions dropdown menu.

   The Tree Structure Audit Result page opens, as shown in the following picture. The table displays a list of validations run against the selected tree structure.
The audit table contains the following columns:

- **Validator Name** - Displays the name of the validator
- **Validation Result** - Displays either a green check mark (success) or a red "X" (failure)
- **Validation Message** - When clicked, displays a validation message and a description
- **Execute Validator** - When clicked, reruns the selected validator.
- **Corrective Action** - When clicked, opens the Edit Tree Structure: Specify Definition page, allowing you to fix a validation error

3. Click **Done** to return to the Tree Structure summary page.

### 20.4 Working with Trees

When you work with trees, you can do any of the following:

- Search
- Create
- Duplicate
- Edit
- Delete

You also can audit trees. For more information, see About Auditing Trees and Tree Versions.

### 20.4.1 How to Search for a Tree

If you wish to duplicate, edit, or delete an existing tree and it is not currently visible in the results list, you can search for it using the following procedure. The procedure assumes that the Tree summary page is open in your web browser.

To search for an existing tree, do the following:

1. In the Search area of the page, construct a search using any or all of the following search criteria:
   - Tree Structure Code
   - Tree Code
• Tree Name

2. Click Search.

All trees matching your search criteria appear in the Results area of the page.

Click Advanced to perform an advanced search by specifying additional options for search. You also can save your search criteria for future use.

20.4.2 How to Create a Tree

The following procedure explains how to create a tree. You will also need to create a tree version with a root node. For more information, see How to Create a Tree Version.

To create a tree, do the following:

1. Click Create, or choose Create Tree from the Actions dropdown menu.

The Create Tree: Specify Definition page opens.

**Figure 20-33  Create Tree: Specify Definition Page**

2. Enter a name for the tree.

3. Enter a code for the tree.

4. Enter a tree-structure name or click the down arrow to select or search for one.

If the tree structure has data sources and parameters defined for it, the Data Source Parameters area also displays, allowing you to edit the parameter values at the tree level.
Note:
Parameter values customized at the tree level will override the default values specified at the tree-structure level.

5. Enter a description of the tree.

6. Enter an image name or click the down arrow to select or search for one.

The image appears in the Preview area.

7. Click Next.

The Create Tree: Specify Labels page displays. The information that appears on the page depends on whether or not a labeling scheme has been selected previously.

*Figure 20-34  Create Tree: Specify Labels - No Labeling Scheme*

If the page shown in Figure 20-34 opens, click Next and skip to Step 11.
If the page shown in Figure 20-35 opens, click Add in the Specify Labels area. The Select and Add: Labels window opens.

**Figure 20-36  Select and Add: Labels Window**

8. Choose a data source from the dropdown list.

9. Select the appropriate available labels and use the arrows to move them back and forth between the Available Labels and Selected Labels areas.

10. Click OK to accept your selections and close the window.

11. Do one of the following:
   - Click Cancel to abort the operation and return to the top-level Manage Trees and Tree Versions page.
   - Click Back to return to the previous page.
   - Click Next to continue to the Create Tree: Specify Access Rules page.

   **Note:**
   The Create Tree: Specify Access Rules page is not yet implemented.

12. Do one of the following:
   - Click Cancel to abort the operation and return to the Manage Trees and Tree Versions page.
   - Click Back to return to the previous page.
   - Click Submit to save the tree without creating a tree version and return to the top-level Manage Trees and Tree Versions page.
   - Click the down arrow next to Submit and select Submit and Add Version to save the tree and begin creating a tree version. For more information, see How to Create a Tree Version.
20.4.3 How to Duplicate a Tree

Duplicating a tree copies only the selected tree. The operation does not copy any of the underlying tree versions.

To duplicate a tree, do the following:

1. Select the tree you wish to duplicate.

   See How to Search for a Tree, if the tree you want to duplicate is not in the current Results list.

2. Click Duplicate, or choose Duplicate from the Actions dropdown menu.

   The Duplicate Tree window opens.

   Figure 20-37  Duplicate Tree Window

3. Enter a duplicate tree code.

4. Do one of the following:

   • Click Cancel to cancel the operation.
   • Click Save and Close to create the duplicate.

5. Click OK to close the Confirmation window.

20.4.4 How to Edit a Tree

When you edit an existing tree, you step through many of the same pages you used to create a tree.

To edit a tree, do the following:

1. Select the tree you wish to edit.

   See How to Search for a Tree, if the tree you want to edit is not in the current Results list.

2. Click Edit, or choose Edit from the Actions dropdown menu.

   The Edit Tree: Specify Definition page opens.

3. Do any of the following:

   • Edit the name of the tree.
   • Edit the description
   • Choose another icon image
   • Edit data-source-parameter values, if this option is available
4. Click **Next**.

The Edit Tree: Specify Labels page opens. The page that displays depends on whether or not the tree structure used while creating the tree has a labeling scheme associated with it.

**Note:**
This procedure assumes that a labeling scheme is present. Skip to Step 6 if the tree you are editing has no labeling scheme associated with it.

5. Do either of the following:
   - Click **Add** to open the Select and Add: Labels window and add a new label. For more information, see *How to Create a Tree*.
   - Select a label and click **Delete** to delete it.

6. Click **Next**.

The Edit Tree: Specify Access Rules page opens.

**Note:**
The Edit Tree: Access Rules page is not yet implemented.

7. Review the data and then do one of the following:
   - Click **Submit** and then click **OK** to close the Confirmation window.
   - Click the arrow to the right of **Submit** and select **Submit and Add Version** from the dropdown list to display the Create Tree Version: Specify Definition page.
     Follow the steps in *How to Create a Tree Version*, to create a new tree version.

### 20.4.5 How to Delete a Tree

When you delete a tree, you also delete the tree versions the tree contains.

To delete a tree, do the following:

1. Select the tree you wish to delete.
   - See *How to Search for a Tree*, if the tree you want to delete is not in the current Results list.

2. Click **Delete**, or choose **Delete** from the **Actions** dropdown menu.

3. Click **Yes** to confirm the deletion.
20.5 Working with Tree Versions

When you work with tree versions, you can do any of the following:

- Search
- Create
- Add nodes to a tree version
- Edit existing nodes
- Drag and drop nodes to move them
- Perform CRUD Operations on Tree Nodes using APIs
- Perform Sub-tree Node Operations using PL/SQL APIs
- Create a new record for a data source
- Duplicate
- Edit
- Delete
- Set tree version status
- Audit tree versions

20.5.1 How to Create a Tree Version

Trees require tree versions. You can create a tree with no tree version, but you must add at least one tree version to the tree after it has been created. You either can create the tree version during the tree-creation process, or by editing an existing tree.

To create a tree version, do the following:

---

**Note:**

This procedure assumes you are editing an existing tree.

---

1. Select the tree to which you want to add a tree version from the list of trees that appears in the Results list.

   See How to Search for a Tree if the tree is not in the current Results list.

2. Do one of the following:

   - Select **Actions > Create Tree Version**.
   - Choose **Create Tree Version** from the **Create** dropdown list

     The Create Tree Version: Specify Definition page opens.
3. Enter a name for the tree version.

4. Enter a description of the tree version.

5. Enter a note, if you have one.

6. Enter an effective start date or click on the calendar icon to select one.

7. Enter an effective end date or click on the calendar icon to select one.

---

**Note:**

Since tree versions are time based, you must select a start date. Selecting an end date is optional.

---

8. Click **Next**.

A tree version with no nodes is created automatically at this point. Procedures for adding nodes to the tree version are described in the following section.

9. Click **OK** to close the Confirmation window.

The Create Tree Version: Specify Nodes page displays.
10. Do one of the following:

- Click **Add** to add a tree node.
- Click **Create** to create a new node in the data source and add it to the hierarchy.

The Add Tree Node window opens.
11. Select a node type:

- **Specific Value** - Indicates that a data source and label will be specified for the node. A data source is required for all labeling schemes. A label is required only if the labeling scheme is set to something other than **None**. If the labeling scheme is **None**, a label is not required. To configure this page's options, see *How to Configure the Add Tree Node: Specific Values*.

- **Values within a range** - Indicates that the node represents a range of values.

**Note:**

This option appears only if **Children Definition: Allow range children** has been selected on the Choose Data Source and Parameters window. If you are adding a root node that you want to specify as range-based node, make sure you have selected **Allow Multiple Root Nodes** for the underlying tree structure on the Create Tree Structure: Data Sources page.
If you select this option, the following window replaces the default Add Tree Node window.

**Figure 20-41  Add Tree Node: Values Within a Range**

![Add Tree Node: Values Within a Range](image)

To configure this page's options, see How to Configure the Add Tree Node: Values Within a Range.

- **Values from referenced hierarchy** - Indicates that the node is a referenced tree node. This option creates a pointer to a tree and node that already exist. Referenced tree nodes do not require a data source and do not allow labeling. If you select this option, the following window replaces the default Add Tree Node window.

**Figure 20-42  Add Tree Node: Referenced Hierarchy**

![Add Tree Node: Referenced Hierarchy](image)

To configure referenced-tree options, see How to Configure the Add Tree Node: Referenced Hierarchy.

12. Click **Save and Close** to add the node(s).

13. Click **Submit** to add the new tree version.

### 20.5.2 About Adding Tree Nodes to a Tree Version

Tree nodes are elements in a tree structure. A tree version must contain at least one root node. If specified, a tree version also can contain multiple root nodes. A node can be the parent of another node if it is one step higher in the hierarchy and closer to the root node.

There are three types of tree nodes:

- Those with specific values
• Those that have a range of values
• Those that have values from a referenced tree

Each type of node has its own configuration options. In addition, you can add tree nodes using a custom Search UI, use drag-and-drop to move nodes once they have been added, and edit existing nodes.

The procedures used to perform these tasks are described in the sections that follow.

### 20.5.2.1 How to Configure the Add Tree Node: Specific Values

The following procedure explains how to configure the Add Tree Node options when the **Specific Value** node type has been selected.

To configure specific values, do the following:

---

**Note:**
This procedure assumes that the Add Tree Node window is open.

---

1. Select a data source.
2. If applicable, select a label.
3. Select an option from the Node Navigator. The navigator enables you to access other available nodes.
4. Select a root node from the Available Nodes list.
5. Click the single Move arrow to move the node to the Selected Nodes area.

---

**Note:**
If the tree structure allows multiple root nodes to be selected, use the single or double Move arrows to move additional nodes.

---

6. Click **Save and Close**.

The Add Tree Node window closes.

The following window shows the root node that has been created. The node name, its description (if any), its label, and the data source details are displayed.
7. Optionally, do one of the following:
   - Highlight the root node and click Add to add a child node, using the same steps listed in this section.
   - Click Create to create a new node in the data source. For more information, see How to Create a Record for a Data Source.

8. Review the data and click Submit.

9. Click OK to close the Confirmation window.

20.5.2.2 How to Configure the Add Tree Node: Values Within a Range

The following procedure explains how to configure the Add Tree Node options when the Values within a range node type has been selected.

To configure values within a range, do the following:

**Note:**

This procedure assumes that the Add Tree Node window is open.

1. Select a data source.

2. Enter a range of values.

3. Click Save and Close.

   The Add Tree Node window closes.
4. Review the data and click Submit.

5. Click OK to close the Confirmation window.

20.5.2.3 How to Configure the Add Tree Node: Referenced Hierarchy

The following procedure explains how to configure the Add Tree Node options when the Values from a referenced hierarchy node type has been selected.

To configure values from a referenced hierarchy, do the following:

**Note:**

This procedure assumes that the Add Tree Node window is open.

This procedure assumes that the Add Tree Node window shown is open.

1. Select a referenced tree.

2. Select a referenced tree version.

**Note:**

The referenced tree and tree version must belong to the same tree structure.

3. Optionally, click the Preview link to ensure you have specified the correct referenced tree version for the selected referenced tree. Confirm the data and click OK to close the window.
4. Click **Save and Close**.

The Add Tree Node window closes and the Create Tree Version: Specify Nodes page refreshes with the referenced-node data.

5. Review the data and click **Submit**.

6. Click **OK** to close the Confirmation window.

**20.5.2.4 About Using Drag-and-Drop to Move Nodes**

Once you have added value-based nodes to a tree version, you can move these nodes around simply by dragging and dropping them.

You can move individual nodes, an entire range of nodes, or an entire referenced node. You cannot, however, move a single node in a range of nodes or a single node in a referenced node.

**20.5.2.5 How to Add a Node Using a Custom Search UI**

If the Search UI is not registered for a data source, the default behavior for all nodes displayed in the Add Tree Node window will be used. However, if you register your own Search UI, it will be used to add and select value nodes instead of the default Search UI.

To add a node using a custom Search UI, do the following:

The procedures for adding a node using a custom Search UI are the same as those found in **About Adding Tree Nodes to a Tree Version**. However, the Add Tree Node window that displays will be the registered custom Search UI rather than the default...
UI. Note that Select Value Nodes has replaced the default Node Navigator, Available Nodes, and Selected Nodes options.

**Figure 20-47  Custom Search UI: Specific Values**

20.5.2.6 How to Edit a Tree Node

You can edit any existing tree node’s details.

To edit a tree node, do the following:

---

**Note:**

This procedure assumes that the Manage Trees and Tree Versions page is open.

---

1. Select a tree version and do one of the following:
   - Click **Edit**.
   - Choose **Edit** from the **Actions** dropdown list.

2. Click **Next** on the Edit Tree Version: Specify Definition page to skip to the Edit Tree Version: Specify Nodes page.

3. Highlight the node you wish to edit and click **Edit**. The Edit Tree Node window.

---

**Note:**

The window opens with the default **Specific value** tree node type selected. You also can edit the node using the other tree node types. For more information, see About Adding Tree Nodes to a Tree Version.
4. Select a data source and click **Edit Node**. The Edit Node window opens.

5. Edit the appropriate details and click **Submit**.

### 20.5.2.7 About Exporting Tree Nodes

You can export all the nodes or selected nodes of a tree version to a CSV file. This export feature helps you in reviewing the tree versions easily, especially when you have a large number of tree nodes.

#### 20.5.2.7.1 How to Export All Nodes in a Tree Version

To export all nodes in a tree version, do the following:

- **Note:**
  
  This procedure assumes that the View Tree Version page is open.

1. Select **Export Nodes** in the **Actions** menu. You can also use the Export Nodes button.
2. All the nodes in a tree version are exported to a CSV file as shown below:

![CSV File for Exporting All Nodes](image)

### 20.5.2.7.2 How to Export the Selected Nodes in a Tree Version

**Note:**

This procedure assumes that the View Tree Version page is open.

1. Select **Export Selected Nodes** in the **Actions** menu.

Select **Export Nodes** in the **Actions** menu. You can also use the Export nodes button.

![Export Selected Nodes in a Tree Version](image)
2. The selected nodes of the selected Tree Version are exported to a CSV file as shown below:

**Figure 20-53 CSV File for Exporting Selected Nodes**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Node Name</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Data Source</td>
</tr>
<tr>
<td>2</td>
<td>Kenneth Walker</td>
<td></td>
<td>Employee Taskflow VO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>MaryBeth Richards</td>
<td></td>
<td>Employee Taskflow VO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Peter Knapp</td>
<td></td>
<td>Employee Taskflow VO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Tracy Peters</td>
<td></td>
<td>Employee Taskflow VO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

20.5.3 How to Create a Record for a Data Source

You can create a record for a data source "on the fly" and add it to the hierarchy. Doing so calls the custom UI you registered with the data source.

You can create a record either when creating a tree version or when editing an existing one.

To create a new data-source record, do the following:

**Note:**

This procedure assumes the following:

- You are editing an existing tree version
- The tree version does not allow multiple root nodes

1. Select the tree version for which you want to create the record and click **Edit**.
2. Click **Next** to access the Edit Tree Version: Specify Nodes window.
3. Highlight an existing node and click **Create**. The Create Tree Node window displays.

**Figure 20-54 Create Tree Node Window**

4. Select a data source and click **Continue**. The Create New Record window opens.

**Note:**

The actual name of the window depends on the node being created.
5. Enter the appropriate information and click **Submit**.
   The Create Tree Node confirmation window displays.

6. Click **OK** to close the window.

### 20.5.4 How to Duplicate a Tree Version

The following procedure explains how to duplicate a tree version.

To duplicate a tree version, do the following:

1. Select a tree version.
2. Click **Duplicate**, or choose **Duplicate** from the **Actions** dropdown menu.
   The Duplicate Tree Version window opens.

3. Enter a duplicate tree version name.
4. Do one of the following:
• Click **Cancel** to cancel the operation.
• Click **Save and Close** to duplicate the tree version.

5. Click **OK** to close the Confirmation window.

### 20.5.5 How to Edit a Tree Version

The following procedure explains how to edit a tree version.

To edit a tree version, do the following:

1. Select a tree version.

2. Click **Edit**, or choose **Edit** from the **Actions** dropdown menu.

   The Edit Tree Version: Specify Definition page opens.

   ![Edit Tree Version: Specify Definition Page](figure20-58)

3. Use the steps in **How to Create a Tree Version** as a guide to editing the tree version.

### 20.5.6 About Performing CRUD Operations on Tree Nodes Using APIs

This section describes the APIs that allow you to perform *Create, Read, Update, and Delete* (CRUD) operations on the Tree Nodes. These APIs do not commit for each CRUD operation, but give you the control to decide whether to commit or not.

The APIs are listed below:

- **add_value_tree_node**

  This API allows you to add a primary key value based tree node.

**Table 20-4 add_value_tree_node API Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Datatype</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_tree_structure_code</td>
<td>Code of the tree structure</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_code</td>
<td>Code of the tree</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>
Table 20-4 (Cont.) *add_value_tree_node* API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Datatype</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_tree_version_id</td>
<td>ID of the tree version</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_parent_tree_node_id</td>
<td>ID of the parent tree node</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_data_source_id</td>
<td>ID of the data source</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk1_value</td>
<td>Value of PK1</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk2_value</td>
<td>Value of PK2</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk3_value</td>
<td>Value of PK3</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk4_value</td>
<td>Value of PK4</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk5_value</td>
<td>Value of PK5</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_label_id</td>
<td>ID of the tree label</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_commit_flag</td>
<td>Commit flag</td>
<td>VARCHAR2</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Default value is Yes. If commit flag is passed as "N", the control to commit is handed over to the API call.

x_tree_node_id | ID of the tree node | VARCHAR2  | Optional          |

- **add_range_tree_node**

This API allows you to add a primary key range based tree node.

Table 20-5  *add_range_tree_node* API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Datatype</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_tree_structure_code</td>
<td>Code of the tree structure</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_code</td>
<td>Code of the tree</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_version_id</td>
<td>ID of the tree version</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_parent_tree_node_id</td>
<td>ID of the parent tree node</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_data_source_id</td>
<td>ID of the data source</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk1_start_value</td>
<td>Start value of PK1</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk2_start_value</td>
<td>Start value of PK2</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk3_start_value</td>
<td>Start value of PK3</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk4_start_value</td>
<td>Start value of PK4</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>
Table 20-5  (Cont.) add_range_tree_node API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Datatype</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_pk5_start_value</td>
<td>Start value of PK5</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk1_end_value</td>
<td>End value of PK1</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk2_end_value</td>
<td>End value of PK2</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk3_end_value</td>
<td>End value of PK3</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk4_end_value</td>
<td>End value of PK4</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk5_end_value</td>
<td>End value of PK5</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_label_id</td>
<td>ID of the tree label</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_commit_flag</td>
<td>Commit flag</td>
<td>VARCHAR2</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Default value is Yes. If commit flag is passed as "N", the control to commit is handed over to the API call.

x_tree_node_id      ID of the tree node  VARCHAR2  Optional

- add_tree_tree_node

This API allows you to add a reference tree-based tree node.

Table 20-6  add_tree_tree_node API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Datatype</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_tree_structure_c</td>
<td>Code of the tree structure</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_code</td>
<td>Code of the tree</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_version_id</td>
<td>ID of the tree version</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_parent_tree_node_id</td>
<td>ID of the parent tree node</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_reference_tree_c</td>
<td>Code of the reference tree</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_reference_tree_v</td>
<td>ID of the reference tree version</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_label_id</td>
<td>ID of the tree label</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_commit_flag</td>
<td>Commit flag</td>
<td>VARCHAR2</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Default value is Yes. If commit flag is passed as "N", the control to commit is handed over to the API call.
Table 20-6  (Cont.) *add_tree_node* API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Datatype</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_tree_node_id</td>
<td>ID of the tree node</td>
<td>VARCHAR2</td>
<td>Optional</td>
</tr>
</tbody>
</table>

*delete_tree_node*

This API allows you to delete a node. All children of the node that is being deleted are promoted as children of its parents node.

Table 20-7  *delete_tree_node* API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Datatype</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_tree_structure_code</td>
<td>Code of the tree structure</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_code</td>
<td>Code of the tree</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_version_id</td>
<td>ID of the tree version</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_node_id</td>
<td>ID of the tree node</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_commit_flag</td>
<td>Commit flag</td>
<td>VARCHAR2</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Default value is Yes.
If commit flag is passed as "N", the control to commit is handed over to the API call.

*update_tree_node*

This API allows you to update a tree node.

Table 20-8  *update_tree_node* API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Datatype</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_tree_structure_code</td>
<td>Code of the tree structure</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_code</td>
<td>Code of the tree</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_version_id</td>
<td>ID of the tree version</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_node_id</td>
<td>ID of the tree node</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_parent_tree_node_id</td>
<td>ID of the parent tree node.</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_data_source_id</td>
<td>ID of the data source</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk1_start_value</td>
<td>Start value of PK1</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>
Table 20-8  (Cont.) update_tree_node API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Datatype</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_pk2_start_value</td>
<td>Start value of PK2</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk3_start_value</td>
<td>Start value of PK3</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk4_start_value</td>
<td>Start value of PK4</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk5_start_value</td>
<td>Start value of PK5</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk1_end_value</td>
<td>End value of PK1</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk2_end_value</td>
<td>End value of PK2</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk3_end_value</td>
<td>End value of PK3</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk4_end_value</td>
<td>End value of PK4</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_pk5_end_value</td>
<td>End value of PK5</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_reference_tree_code</td>
<td>Code of the reference tree</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_reference_tree_version_id</td>
<td>ID of the reference tree version</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_label_id</td>
<td>ID of the tree label</td>
<td>VARCHAR2</td>
<td>Optional</td>
</tr>
<tr>
<td>p_commit_flag</td>
<td>Commit flag</td>
<td>VARCHAR2</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Default value is Yes.
If commit flag is passed as "N", the control to commit is handed over to the API call.

• move_tree_node
This API allows you to move a tree node under a new parent.

Table 20-9  move_tree_node API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Datatype</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_tree_structure_code</td>
<td>Code of the tree structure</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_code</td>
<td>Code of the tree</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_version_id</td>
<td>ID of the tree version</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_node_id</td>
<td>ID of the tree node</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_dest_parent_tree_node_id</td>
<td>ID of the destination parent tree node</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>
Table 20-9  (Cont.) move_tree_node API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Datatype</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_commit_flag</td>
<td>Commit flag</td>
<td>VARCHAR2</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Default value is Yes. If commit flag is passed as &quot;N&quot;, the control to commit is handed over to the API call.</td>
</tr>
</tbody>
</table>

20.5.7 About Performing Sub-tree Node Operations Using PL/SQL APIs

PL/SQL APIs allow you to move, copy, and remove sub-tree nodes. The following are the PL/SQL APIs that allow you to perform the operations:

- **move_sub_tree_node**

  This API allows you to move a given node along with its descendants from one hierarchy version to another hierarchy or within the same hierarchy tree version. The destination hierarchy can belong to same tree code or a different tree code belonging to same tree structure. When move_sub_tree_node operation happens across the hierarchy or within the same hierarchy, flattening logs will be reset to ensure that next run of flattening generates accurate flattening data. A tree node id pointing to details of new subtree in destination hierarchy will be returned.

Table 20-10  move_sub_tree_node API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Datatype</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_tree_structure_code</td>
<td>Code of the tree structure</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_code</td>
<td>Code of the tree</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_version_id</td>
<td>ID of the tree version</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_node_id</td>
<td>ID of the tree node</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_dest_parent_tree_node_id</td>
<td>ID of the destination parent tree node</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_dest_tree_version_id</td>
<td>ID of the destination parent tree version</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_dest_tree_code</td>
<td>Destination tree code</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_commit_flag</td>
<td>Commit flag</td>
<td>VARCHAR2</td>
<td>Optional</td>
</tr>
<tr>
<td>x_tree_node_id</td>
<td>ID of the tree node</td>
<td>VARCHAR2</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Default value is Yes. If commit flag is passed as &quot;N&quot;, the control to commit is handed over to the API call.</td>
</tr>
</tbody>
</table>

- **copy_sub_tree_node**
This API allows you to copy a given node along with its descendants within the same hierarchy tree version. The tree version must allow duplicates for the copy operation to be successful. The destination hierarchy can belong to same tree code or a different tree code belonging to same tree structure. When copy_sub_tree_node operation happens, flattening logs will be reset to ensure that next run of flattening generates accurate flattening data. A tree node id pointing to details of new subtree in destination hierarchy will be returned.

**Table 20-11  copy_sub_tree_node API Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Datatype</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_tree_structure_code</td>
<td>Code of the tree structure</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_code</td>
<td>Code of the tree</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_version_id</td>
<td>ID of the tree version</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_node_id</td>
<td>ID of the tree node</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_dest_parent_tree_node_id</td>
<td>ID of the destination parent tree node</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_dest_tree_version_id</td>
<td>ID of the destination parent tree version</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_dest_tree_code</td>
<td>Destination tree code</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_commit_flag</td>
<td>Commit flag</td>
<td>VARCHAR2</td>
<td>Optional</td>
</tr>
<tr>
<td></td>
<td>Default value is Yes.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>If commit flag is passed as “N”, the</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>control to commit is handed over to the</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>API call.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x_tree_node_id</td>
<td>ID of the tree node</td>
<td>VARCHAR2</td>
<td>Optional</td>
</tr>
</tbody>
</table>

- **remove_sub_tree_node**

This API allows you to remove a given node along with its descendants from any hierarchy. When remove_sub_tree_node operation happens, all the descendants will be deleted.

**Table 20-12  remove_sub_tree_node API Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Datatype</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_tree_structure_code</td>
<td>Code of the tree structure</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_code</td>
<td>Code of the tree</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_version_id</td>
<td>ID of the tree version</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
<tr>
<td>p_tree_node_id</td>
<td>ID of the tree node</td>
<td>VARCHAR2</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>
Table 20-12  (Cont.) remove_sub_tree_node API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Datatype</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_commit_flag</td>
<td>Commit flag</td>
<td>VARCHAR2</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Default value is Yes.
If commit flag is passed as "N", the control to commit is handed over to the API call.

20.5.8 How to Set Tree Version Status

Although trees do not have status, tree versions do. You can set tree version status to any one of the following:

- Draft
- Active
- Inactive

To activate a tree version, the tree version’s tree structure must already be in Active status.

Setting a tree version’s status to Active automatically triggers an audit of that tree structure.

To set the status of a tree version, do the following:

1. Select a tree version.
2. Choose the appropriate status option from the Actions > Set Status dropdown menu.
3. Click OK to close the Confirmation window.

20.5.9 About Auditing Trees and Tree Versions

Auditing tree and tree version data verifies that it conforms to all rules and ensures data integrity. Running audits allow you to view audit details and messages, and to correct any validation errors that the audit detects. There are three ways to run an audit:

- Run an immediate audit
- Schedule an audit
- Trigger an audit through a service API

The following table describes what each validator checks for, as well as possible reasons why each validator might fail.
## Table 20-13  Validator Descriptions

<table>
<thead>
<tr>
<th>Validator</th>
<th>Checks for...</th>
<th>Validation may have failed because...</th>
<th>To correct...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effective Date Validator</td>
<td>The effective start and end dates of the tree version should be valid.</td>
<td>Effective end date is set to a value that is not greater than effective start date.</td>
<td>Modify the effective start and/or end dates so that effective start date falls before effective end date.</td>
</tr>
<tr>
<td>Root Node Validator</td>
<td>If Allow Multiple Root Nodes flag on the tree structure has been set to No, the tree version must contain exactly one root node if it is not empty. If the flag has been set to Yes, this restriction does not apply.</td>
<td>Allow Multiple Root Nodes flag has been set to No at the tree structure, but the tree version has multiple root nodes.</td>
<td>Modify the tree version so that there is exactly one root node.</td>
</tr>
<tr>
<td>Data Source Max Depth Validator</td>
<td>For each data source in the tree structure, if the data source is depth-limited, the data in the tree version must adhere to the specified depth limit. This restriction does not apply to data sources that have no depth restriction (depth = -1 means unlimited depth).</td>
<td>Tree version has data at a depth greater than the specified depth limit on one or more data sources.</td>
<td>Modify the tree version so that all nodes are at a depth that complies with the data source depth limit.</td>
</tr>
<tr>
<td>Duplicate Node Validator</td>
<td>If Allow Duplicate Nodes flag on the data source has been set to No, the tree version should not contain more than one node with the same primary key from the data source. If the flag has been set to Yes, duplicate nodes are permitted.</td>
<td>Your tree version contains duplicate nodes with the same primary key.</td>
<td>Remove any duplicate nodes from the tree version.</td>
</tr>
<tr>
<td>Validator</td>
<td>Checks for...</td>
<td>Validation may have failed because...</td>
<td>To correct...</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Available Node Validator</td>
<td>All the nodes in the tree version should be valid and available in the underlying data source.</td>
<td>• A node in the tree version does not exist in the data source. Deletion of data items from the data source without removing the corresponding nodes from the tree version can result in orphaned nodes in the tree version. For example, if you have added node A into your tree version, but node A was subsequently deleted from your data source but not from the tree version, it will fail this validation. • Your tree version contains a tree reference node, which references another tree version that does not exist.</td>
<td>• Remove any orphaned nodes from the tree version. • Correct any tree reference nodes so they reference existing tree versions.</td>
</tr>
<tr>
<td>Node Relationship Validator</td>
<td>All nodes should adhere to the relationships mandated by the data sources registered in the tree structure.</td>
<td>The tree structure has data sources arranged in a parent-child relationship, but the nodes in the tree do not adhere to the same parent-child relationship. For example, if the tree structure has a Project data source with a Task data source as its child, Task nodes should always be under Project nodes in the tree version. This validator will fail if there are instances where a Project node has been added as a child of a Task node.</td>
<td>Modify the tree version so that the nodes adhere to the same parent-child relationships as the data sources.</td>
</tr>
<tr>
<td>SetID Restricted Node Validator</td>
<td>For each data source that has Restrict Tree Node List of Values Based on SetID flag set to Yes, for each tree node, the underlying node in the data source must belong to the same set as the tree itself. This restriction does not apply when the flag is set to No.</td>
<td>The data source has Restrict Tree Node List of Values Based on SetID flag set to Yes, but the tree version has nodes whose data source values belong to a different set than the tree.</td>
<td>Modify the tree version so that all nodes in the tree have data sources with SetID matching that of the tree.</td>
</tr>
</tbody>
</table>
## Table 20-13  (Cont.) Validator Descriptions

<table>
<thead>
<tr>
<th>Validator</th>
<th>Checks for...</th>
<th>Validation may have failed because...</th>
<th>To correct...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label Enabled Node Validator</td>
<td>If the tree structure has a Labeling Scheme specified, all nodes should have labels. This restriction does not apply when the Labeling Scheme is set to None.</td>
<td>The tree structure has a labeling scheme but the tree version has nodes without labels.</td>
<td>Assign labels to any nodes that do not have labels.</td>
</tr>
<tr>
<td>Date Restricted Node Validator</td>
<td>If Restrict Tree Node List of Values Based on Date Range flag on the tree structure has been set to Yes, each node in the underlying data source must have date effectivity during the date effectivity range of the tree version. If the flag is set to No, this restriction does not apply.</td>
<td>Restrict Tree Node List of Values Based on Date Range flag has been set to Y, but there are data source nodes that are not have effective during the tree version's effective date range. For example, if the tree version is effective from Jan-01-2012 to Dec-31-2012, all nodes in the tree version must be effective from Jan-01-2012 to Dec-31-2012 at a minimum. It is acceptable for the nodes to be effective for a date range that exceeds the tree version's effective date range (for example, the node data source value is effective from Dec-01-2011 to Mar-31-2012). It is not acceptable if the nodes are effective for none or only part of the tree version's effective date range (for example, the node data source value are only effective from Jan-01-2012 to June-30-2012).</td>
<td>Ensure that for all nodes in the tree version, they have date effectiveness at least for the effective date range for the tree version.</td>
</tr>
<tr>
<td>Multiple Active Tree Version Validator</td>
<td>If Allow Multiple Active Tree Flags on the tree structure has been set to No, there should not be more than one active tree version under a tree at any time. If Allow Multiple Active Tree Versions is set to No, this restriction does not apply.</td>
<td>Allow Multiple Active Tree Versions has been set to N but there is more than one active tree version in the tree for the same date range.</td>
<td>Make no more than one tree version Active within the same date range and set the others to Inactive or Draft.</td>
</tr>
</tbody>
</table>
Table 20-13  (Cont.) Validator Descriptions

<table>
<thead>
<tr>
<th>Validator</th>
<th>Checks for...</th>
<th>Validation may have failed because...</th>
<th>To correct...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range Based Node</td>
<td>If Allow Range Children on the data source has been set to No, range-based nodes are not permitted from that data source. If the flag is set to Yes, this restriction does not apply.</td>
<td>There are range-based nodes from a data source that has Allow Range Children set to N.</td>
<td>Ensure that any range nodes in your tree version are from a data source that has Allow Range Children set to Y.</td>
</tr>
<tr>
<td>Terminal Node Validator</td>
<td>If Allow Usage as Leaves flag is set to N at the data source level, values from that data source cannot be added as leaves (terminal nodes) to the tree version. If Allow Usage as Leaves flag is set to Y, this restriction does not apply.</td>
<td>There are leaf nodes (terminal nodes) whose values come from a data source marked with Allow Usage as Leaves flag set to N. Only data sources with Allow Usage as Leaves set to Y can have their values used as leaves.</td>
<td>Modify the tree version so that all terminal nodes come from data sources with Allow Usage as Leaves set to Y.</td>
</tr>
<tr>
<td>Usage Limit Validator</td>
<td>If Usage Limit is set to Use All Values at the data source level, every value in the data source must appear as a node in the tree. If Usage Limit is set to None, this restriction does not apply.</td>
<td>The data source has Usage Limit set to Use All Values, but there are values in the data source that are not in the tree version.</td>
<td>Add nodes to the tree version for each data source value that is not yet present.</td>
</tr>
</tbody>
</table>

20.5.9.1 How to Run an Immediate Audit

To run an immediate audit, do the following:

Note:

The following procedure assumes the Manage Trees and Tree Versions page is open.

1. Select the tree or tree version you wish to audit.

   Selecting a tree runs an audit on all tree versions in that tree. Selecting a specific tree version runs an audit only on that tree version.

2. Choose Audit from the Actions dropdown menu.

   The audit runs and the Trees Audit Result page opens.
Figure 20-59  Trees Audit Result Page

The page contains two sections:

- **Audit Results** - displays a list of all previously run audits
- **Validation Details** - displays validation results and messages and allows you correct validation errors

The Audit Results section contains the following columns:

- **Audit Request** - displays the audit request ID number
- **Tree Version Name** - displays the name of the tree version
- **Audit Result** - displays either a green check mark (success) or a red "X" (failure)
- **Tree Structure Code** - displays the tree structure code
- **Tree Code** - displays the tree code
- **Start Time** - displays the date and time the audit began
- **End Time** - displays the date and time the audit was completed

The Validation Details section contains the following columns:

- **Name** - displays the names of the tree or tree version and the audit validators
- **Validation Result** - displays either a green check mark (success) or a red "X" (failure)
- **Validation Message** - when clicked, displays a validation message and a description
- **Corrective Action** - when clicked, opens the appropriate trees application task flow page, allowing you to fix a validation error

### 20.5.9.2 How to Schedule an Audit

To schedule an audit, do the following:

1. Click **Schedule Audit**.
   
   The Schedule Audit window opens.
2. Do any of the following:

- Configure the basic options.
- Click **Process Options** to configure language, territory, timezone, and other options.
- Click **Advanced** to configure detailed schedule, output, and notification options.

3. Click **Submit** and then click **OK** to close the confirmation dialog.

### 20.5.9.3 How to Trigger an Audit Through a Service API

To trigger an audit, through a service API, use the service API shown in the example that follows.

**Example 20-1 Audit Service API**

```java
/**
 * Processes the audit scheduled for any tree or tree version.
 * @param requestId Request ID for scheduled audit (for online can be defaulted as -1)
 * @param auditType Auditing mode where audit invoked for tree/tree version (pass as TREE_AUDIT for tree and TREE_VERSION_AUDIT for tree version).
 * @param tsCode Tree Structure Code
 * @param treeCode Tree Code
 * @param treeVersionId Tree Version Id
 */
public void processAudit(Long requestId, String auditType, String tsCode, String treeCode, String treeVersionId)
```

### 20.5.10 About Flattening Rows and Columns

The tree-flattening process filters an implicit tree structure into a simple sequence of leaves. It coalesces nodes so that each sub-tree has a single cache list representing all of its children at one transformed level.
Tables that store flattened data are either row or column flattened. By eliminating recursive queries, row flattening is particularly useful for efficiently performing operations across an entire sub-tree.

**Understanding Row Flattening**

Row flattening is a technique where parent-child information is optimized for runtime performance by storing additional rows in a table (as compared to just normalized parent-child rows) to instantly find all descendants to a parent value, without initiating a Connect By SQL statement.

Normalized data, for example, might be the following:

**Corporation - Sales Division**

**Sales Division - Region**

In a row flattened table, the above rows are still stored but one additional row is added:

**Corporation - Region**

In addition, additional columns are added to store the “depth” from top parent.

**Understanding Column Flattening**

Column flattening is a technique where parent-child information is optimized for runtime performance by storing additional column in a table for all parents of a child.

Normalized data, for example, might be the following:

**Corporation - Sales Division**

**Sales Division - Region**

**Sales Division**

In a column-flattened table, the above data is converted to rows and columns.

<table>
<thead>
<tr>
<th>Table 20-14</th>
<th>Rows and Columns in a Column-Flattened Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column 1</td>
<td>Column 2</td>
</tr>
<tr>
<td>Region</td>
<td>Sales Division</td>
</tr>
</tbody>
</table>

Usually, the number of levels (possible parents) are pre-defined to a maximum number and may also have additional "dummy" values on levels where real values are missing.

**20.5.10.1 How to Flatten a Row or Column**

The following procedure assumes the Manage Trees and Tree Versions page is open.

1. Select the tree version you want to flatten.
2. Choose **Column Flattening** or **Row Flattening** from the **Actions** dropdown menu.

The Flattening page opens.

*Figure 20-62  Schedule Flattening Page*

3. Click **Schedule Flattening**.

4. The **Schedule Flattening** page opens. Configure the options and click **Submit**.

5. Click **Online Flattening** or **Force Flattening**. Either of these actions will flatten the row.

6. Click **OK** to close the confirmation window.

*Figure 20-63  Flattening Confirmation*

7. Click **Done** to return to the Manage Trees and Tree Versions page.

### 20.6 Manage Labels in the Generic Label Data Source

When a label is chosen for a tree structure, the label data source can be either of the following:

- A custom, product-team-owned label data source
- The generic data label source.

This section describes how to use the **Central Labels** tab of the trees application launch page to create, edit, and delete values in the generic label data source.

All procedures assume that the Manage Labels summary page is open in your web browser.

#### 20.6.1 How to Search for a Label

If you wish to edit or delete an existing label and it is not currently visible in the results list, you can search for it using the following procedure.

**To search for a label:**

1. In the Search area of the page, construct a search using any or all of the following search criteria:
   - Tree Structure Code
   - Name
2. Click **Search**.

   All labels matching your search criteria appear in the Results area of the page. Click **Advanced** to perform an advanced search by specifying additional options. You also can save your search criteria for future use.

### 20.6.2 How to Create a Label

The following procedure explains how to create a new label.

**To create a label:**

1. From the Manage Labels summary page, click **Create**, or choose **Create Label** from the **Actions** dropdown menu.

   The Create Label page displays An additional field Set Name will appear if it is used by the selected tree structure.

   ![Create Label Page](image)

2. Enter a tree structure code or click the down arrow to select or search for one.

   The page refreshes and the **Data Source** field populates with an appropriate value.

3. Enter a short name for the label.

4. Enter a name for the label.

5. If you wish, enter a description.

6. If you wish, enter an icon name.

7. Enter an effective start date, or click the calendar icon to select one.

8. Enter an effective end date, or click the calendar icon to select one.

9. Enter a Set Name or click the down arrow to select or search one.
10. Click **Save and Close** to create the label.

11. Click **OK** to close the Confirmation window.

The Manage Labels summary page displays showing the new label in the Results list.

### 20.6.3 How to Edit a Label

The following procedure explains how to edit an existing label.

**To edit a label:**

1. Select the label you want to edit.

   See **How to Search for a Label**, if the label is not in the current Results list.

2. From the Manage Labels summary page, click **Edit**, or choose **Edit** from the **Actions** dropdown menu.

   The Edit Label page displays.

![Figure 20-65   Edit Label Page](image)

3. Edit the appropriate data.

4. Do one of the following:
   - Click **Save and Close** to save your changes and exit the editing session.
   - Click **Cancel** to cancel the operation.

### 20.6.4 How to Delete a Label

The following procedure explains how to delete an existing label.

**To delete a label:**

1. Select the label you want to delete.

   See **How to Search for a Label**, if the label is not in the current Results list.
2. From the Manage Labels summary page, click **Delete**, or choose **Delete** from the **Actions** dropdown menu.

The warning page displays.

*Figure 20-66  Delete Label Warning Page*

![Delete Label Warning Page](image)

3. Do one of the following:
   
   - Click **No** to cancel the operation.
   
   - Click **Yes** to delete the label.

20.7 About Using the Applications Hierarchy Component to Develop Applications

Now that you have worked with tree structures, trees, and tree versions, you can start developing applications in JDeveloper using the Applications Hierarchy component.

The Applications Hierarchy component is denoted by the **fnd:hierarchy** tag and contains two facets: **hierarchy** and **toolbar**. The **hierarchy** facet holds the **af:tree** or **af:treeTable**; the **toolbar** facet can hold action buttons used with items within the **tree** or **treeTable**.

The following picture shows an example of the Applications Hierarchy component in JDeveloper.

*Figure 20-67  Applications Hierarchy Component*

![Applications Hierarchy Component](image)

You can add any JSF or ADF Faces component to these facets, even with the generated **af:tree** or **af:treeTable**. The **fnd:hierarchy** tag supports the TreeCode and TreeVersionId properties to display specific trees or tree versions.

You can create two types of Hierarchy applications: Tree and Tree Table.

20.7.1 How to Create a Tree Application

The following section explains how to create a tree application using the Applications Hierarchy component.
Before you begin:
Create an application initialized for use with Oracle Middleware Extensions for Applications. For more information, see Set Up Your Development Environment.

To create a Tree application, do the following:

1. Create a new JSF/JSPX page.
2. From the ADF Faces page in the Component Palette, select Applications.
3. From the Applications page in the Component Palette, select Hierarchy and drag it to your .jspx file's visual editor.

The Initialize Applications Connection window opens:

4. Choose a connection from the dropdown menu and click OK.
If there is no existing connection, click Add to create a new one.
A second Initialize Applications Connection window opens.

5. Choose a tree structure from the dropdown list.
6. Select Tree.
7. Select Read-only if you want the application to be read only.
8. Click OK.

The tree appears in the visual editor.
9. Do one of the following:
   • Run the .jspx file from the Application Navigator.
   • Run the .jspx file from the visual editor.
   A browser window opens and the application runs.

20.7.2 How to Create a Tree Table Application

   The following section explains how to create a tree table application using the applications Hierarchy component.

   **Before you begin:**

   Create an application initialized for use with Oracle Middleware Extensions for Applications. For more information, see Set Up Your Development Environment.

   To create a Tree Table application, do the following:

   1. Follow Steps 1 through 5 in How to Create a Tree Application.

   2. Select Tree Table.

      The Initialize Applications Connection window redisplayes with additional fields.
3. Select Read-only if you want the application to be read only.
4. Select the available attributes to display at each tree level.
5. Click OK.

The tree table appears in the visual editor.

6. Run the .jspx from either the Application Navigator or the visual editor.

A browser window opens and the application runs.
20.7.3 How to Sort Hierarchies

The Sort option allows you to sort tree-node hierarchies in ascending or descending order, and is enabled by default.

1. Select a column.
2. Navigate to View > Sort.
3. Select Ascending or Descending.

20.8 About Integrating Custom Task Flows into the Applications Hierarchy Component

The Applications Hierarchy component supports six node operations – Add, Create, Duplicate, Edit, Remove, and Delete – which are performed in a standard popup window in the user interface. Since all nodes in a tree come from their data sources, you must create a custom task flow for each operation that requires one, and register it in your data source before you can use it.

Node operations that only manage data in the tree table do not require custom task flows. For example, adding a node affects only the data in the tree table. However, searching for a node requires access to the data source. Subsequently, you must create and register a custom task flow that will enable node searches.

In addition, since removing a node affects only the data in the tree table, you do not need to create and register a custom task flow for this operation.
Related Links
The following document provides more information about the topic discussed in this section:

- For more information about task flows, see Getting Started with ADF Task Flows in the Developing Fusion Web Applications with Oracle Application Development Framework.

20.8.1 About Registering Custom Task Flows
You use the Property Inspector in Oracle JDeveloper to register custom task flows.

Figure 20-76  Data Source Properties

The full code used to register each custom task flow in the data-source view object is shown in the examples that follow.

Example 20-2  Search Task Flow for the Add Node Operation

```xml
<Properties>
  <SchemaBasedProperties>
    <fnd:SEARCH_PAGE Value="/WEB-INF/EmpSearch.xml#EmpSearchTF"/>
    ...
    ...
  </SchemaBasedProperties>
</Properties>
```

Example 20-3  Create Task Flow

```xml
<Properties>
  <SchemaBasedProperties>
    <fnd:CREATE_PAGE Value="/WEB-INF/CreateEmpNode.xml#CreateEmpNode"/>
    ...
    ...
  </SchemaBasedProperties>
</Properties>
```

Example 20-4  Duplicate Task Flow

```xml
<Properties>
  <SchemaBasedProperties>
    <fnd:DUPLICATE_PAGE Value="/WEB-INF/DupEmpNode.xml#DupEmpNode"/>
    ...
  </SchemaBasedProperties>
</Properties>
```
20.8.2 About Creating Custom Task Flows

This section discusses how to create custom task flows for the Search, Create, Duplicate, Edit, and Delete node operations.

20.8.2.1 How to Create a Search Task Flow for the Add Node Operation

The Search task flow provides a shortcut to select nodes. In the task flow, you specify the Search page fragment, task-flow parameters, back-end Java bean, and task-flow activities.

To create the Search task flow, do the following:

1. Define a task-flow parameter, searchHierParamBean, to pass values between the Hierarchy component and the registered Search task flow in pageFlowScope.

   ```xml
   <input-parameter-definition>
   <name>searchHierParam</name>
   <value>${pageFlowScope.searchHierParam}</value>
   <class>oracle.apps.fnd.applcore.trees.ui.managed.HierParamBean</class>
   </input-parameter-definition>
   ```

2. Define Search task-flow return activities and use them as the ends of the task flow. For example, to define the Submit and Cancel task-flow return activities:

   ```xml
   <task-flow-return id="Submit">
   <outcome>
   <name>Submit</name>
   </outcome>
   </task-flow-return/>

   <task-flow-return id="Cancel">
   <outcome>
   <name>Cancel</name>
   </outcome>
   </task-flow-return/>
   ```

   Normally, Cancel is a free event. However, you must pass values back when Submit is triggered. Therefore, you can have a submit() ActionListener mapping to it.

3. In your back-end bean, use the following code to get the task-flow parameter:
import oracle.apps.fnd.applcore.trees.ui.managed.HierParamBean;
...

HierParamBean searchParam = (HierParamBean) AdfFacesContext.getCurrentInstance().getPageFlowScope().get("searchHierParam");
...

For the Search task flow, you do not need any input parameters from the Hierarchy component. The only thing this task flow does is to return the search results. Therefore, after looking up tree nodes in Search task flow, you must pass back the primary keys of the selected nodes by calling the HierParamBean method setSelectedNodes(List) when the Submit return activity is invoked:

public void submit(ActionEvent event) {
    // The inner list is the list of primary keys for each node. Trees supports up to five primary keys.
    // The outer list is the list of selected nodes
    List<List<String>> nodes = ...
    ...
    searchParam.setSelectedNodes(nodes);
}

4. In the page definition file of the page using the Hierarchy component, add the following task-flow entry in the "executables" section:

    <taskflow id="searchTaskflow"
        taskFlowId="#{backingBeanScope.AddNodeBean.searchTaskflow}"
        activation="deferred"
        Refresh="ifNeeded"
        xmlns="http://xmlns.oracle.com/adf/controller/binding">
        <parameters>
            <parameter id="searchHierParam" xmlns="http://xmlns.oracle.com/adfm/uimodel"
                value="#{pageFlowScope.searchHierParam}"/>
        </parameters>
    </taskflow>

5. A page definition file is created at design time for each JSF/jspx file created, with the following taskflows created in it under the "executables" section:

    </taskFlow>
    </taskFlow>
    <taskFlow id="searchTaskflow"
        taskFlowId="#{backingBeanScope.AddNodeBean.searchTaskflow}"
        activation="deferred"
        Refresh="ifNeeded"
        xmlns="http://xmlns.oracle.com/adf/controller/binding">
        <parameters>
            <parameter id="searchHierParam" xmlns="http://xmlns.oracle.com/adfm/uimodel"
                value="#{pageFlowScope.searchHierParam}"/>
        </parameters>
    </taskFlow>
20.8.2.2 How to Create a Create Task Flow

The Create task flow is used to create a new node in the data-source table. In the task flow, you specify the Create page fragment, task-flow parameters, back-end Java bean, and task-flow activities.

To create the Create task flow, do the following:

1. Define a task-flow parameter, createHierParamBean, to pass values between the Hierarchy component and the registered Create task flow in pageFlowScope.

   ```xml
   <input-parameter-definition>
   <name>createHierParam</name>
   <value>${pageFlowScope.createHierParam}</value>
   <class>oracle.apps.fnd.appcore.trees.ui.managed.HierParamBean</class>
   </input-parameter-definition>
   ```

2. To capture the event of dismissing the Create popup window, define Create task-flow return activities and use them as the ends of the task flow. For example, to define the Submit and Cancel task-flow return activities:

   ```xml
   <task-flow-return id="Submit" />
   <outcome>
   <name>Submit</name>
   </outcome>
   </task-flow-return>

   <task-flow-return id="Cancel" />
   <outcome>
   <name>Cancel</name>
   ```
Normally, Cancel is a free event. However, you must pass values back when Submit is triggered. Therefore, you can have a submit() actionListener mapping to it.

3. In your back-end bean, use the following code to get the task-flow parameter:

```java
import oracle.apps.fnd.applcore.trees.ui.managed.HierParamBean;
...
HierParamBean createParam =
(HierParamBean)AdfFacesContext.getCurrentInstance().getPageFlowScope().get
("createHierParam");
...
```

For the Create task flow, you do not need any input parameters from the Hierarchy component. The only thing this task flow does is to return the new node. Therefore, you must pass back its primary keys by calling the HierParamBean method setNewPkValue() after creating the node. The Hierarchy component will get the new node's primary keys after the Create task-flow is dismissed. For example, in the submit() actionListener that maps to the Submit return activity, you pass new primary keys back:

```java
public void submit(ActionEvent event) {
    List<String> newPk = ...
    ...
    createParam.setNewPkValue(newPk);
}
```

4. In the page definition file of the page using the Hierarchy component, add the following task-flow entry in the "executables" section:

```xml
<taskFlow id="createTaskflow"
    taskFlowId="#{backingBeanScope.CreateNodeBean.createTaskflow}"
    activation="deferred"
    Refresh="ifNeeded"
    xmlns="http://xmlns.oracle.com/adf/controller/binding">
    <parameters>
        <parameter id="createHierParam" xmlns="http://xmlns.oracle.com/adfm/uimodel"
            value="#{pageFlowScope.createHierParam}"/>
    </parameters>
</taskFlow>
```

20.8.2.3 How to Create a Duplicate Task Flow

The Duplicate task flow is used to duplicate a node in the data-source table. In the task flow, you specify the Duplicate page fragment, task-flow parameters, back-end Java bean, and task-flow activities.

To create a Duplicate task flow, do the following:

1. Define a task-flow parameter, dupHierParamBean, to pass values between the Hierarchy component and the registered Duplicate task flow in pageFlowScope.

```xml
<input-parameter-definition>
    <name>dupHierParam</name>
    <value>#{pageFlowScope.dupHierParam}</value>
    <class>oracle.apps.fnd.applcore.trees.ui.managed.HierParamBean</class>
</input-parameter-definition>
```
2. To capture the event of dismissing the Duplicate popup window, define Duplicate task-flow return activities and use them as the ends of the task flow. For example, to define the Submit and Cancel task-flow return activities:

```xml
<task-flow-return id="Submit">
  <outcome>
    <name>Submit</name>
  </outcome>
</task-flow-return/>

<task-flow-return id="Cancel">
  <outcome>
    <name>Cancel</name>
  </outcome>
</task-flow-return/>
```

Normally, Cancel is a free event. However, you must pass values back when Submit is triggered. Therefore, you can have a submit() actionListener mapping to it.

3. In your back-end bean, using the following code to get the task-flow parameter:

```java
import oracle.apps.fnd.applcore.trees.ui.managed.HierParamBean;
...

HierParamBean dupParam =
(HierParamBean)AdfFacesContext.getCurrentInstance().getPageFlowScope().get("dupHierParam");
...
```

For the Duplicate task flow, you need to know the selected node from the Hierarchy component. In the Duplicate popup window, the selected node’s attributes are shown by default so that users can create another tree node. After creating a new node, you pass back its primary key. You can perform all of these tasks by calling HierParamBean methods.

For example, in the task-flow initializer, you can use the following code to get the primary keys of the selected node:

```java
List<String> pkValue = dupParam.getPkValue();
```

In the submit() actionListener that maps to the Submit return activity, you pass the new primary keys back:

```java
public void submit(ActionEvent event) {
  List<String> newPk = ...
  ...
  dupParam.setNewPkValue(newPk);
}
```

4. In the page definition file of the page using the Hierarchy component, add the following task-flow entry in the "executables" section:

```xml
<taskflow id="dupTaskflow" 
  taskFlowId="#{backingBeanScope.DupNodeBean.dupTaskflow}" 
  activation="deferred" 
  Refresh="ifNeeded" 
  xmlns="http://xmlns.oracle.com/adf/controller/binding">
  <parameters>
    <parameter id="dupHierParam" xmlns="http://xmlns.oracle.com/adfm/uimodel" 
      value="#{pageFlowScope.dupHierParam}"/>
```
20.8.2.4 How to Create an Edit Task Flow

The Edit task flow is used to edit an existing node in the data-source table. In the task flow, you specify the Edit page fragment, task-flow parameters, back-end Java bean, and task-flow activities.

To create the Edit task flow, do the following:

1. Define a task-flow parameter, editHierParamBean, to pass values between the Hierarchy component and the registered Edit task flow in pageFlowScope.

   ```xml
   <input-parameter-definition>
   <name>editHierParam</name>
   <value>${pageFlowScope.editHierParam}</value>
   <class>oracle.apps.fnd.applcore.trees.ui.managed.HierParamBean</class>
   </input-parameter-definition>
   ```

2. To capture the event of dismissing the Edit popup window, define Edit task-flow return activities and use them as the ends of the task flow. For example, to define the Submit and Cancel task-flow return activities:

   ```xml
   <task-flow-return id="Submit">
   <outcome>
   <name>Submit</name>
   </outcome>
   </task-flow-return/>
   
   <task-flow-return id="Cancel">
   <outcome>
   <name>Cancel</name>
   </outcome>
   </task-flow-return/>
   ```

   Normally, Cancel is a free event. However, you must pass values back when Submit is triggered. Therefore, you can have a submit() actionListener mapping to it.

3. In your back-end bean, using the following code to get the task-flow parameter:

   ```java
   import oracle.apps.fnd.applcore.trees.ui.managed.HierParamBean;
   ...
   
   HierParamBean editParam = (HierParamBean)AdfFacesContext.getCurrentInstance().getPageFlowScope().get("editHierParam");
   ...
   ```

   For the Edit task flow, you need to know the current node from the Hierarchy component. In the Edit popup window, the current node's attributes are shown by default so that users can update the tree node. After updating the node, you must indicate whether or not the update was successful. You can perform all of these tasks by calling HierParamBean methods.

   For example, in the task-flow initializer, you can use the following code to get the primary keys of the selected node:

   ```java
   List<String> pkValue = editParam.getPkValue();
   ```

   In the submit() actionListener that maps to the Submit return activity, you specify the result:
public void submit(ActionEvent event) {
    ...
    editParam.setUpdated(true);
}

4. In the page definition file of the page using the Hierarchy component, add the following task-flow entry in the "executables" section:

```
<taskflow id="editTaskflow"
    taskFlowId="#{backingBeanScope.EditNodeBean.editTaskflow}"
    activation="deferred"
    Refresh="ifNeeded"
    xmlns="http://xmlns.oracle.com/adf/controller/binding">
    <parameters>
        <parameter id="dupHierParam" xmlns="http://xmlns.oracle.com/adfm/uimodel"
            value="#{pageFlowScope.editHierParam}"/>
    </parameters>
</taskflow>
```

20.8.2.5 How to Create a Delete Task Flow

The Delete task flow is used to delete an existing node in the data-source table. In the task flow, you specify the Delete page fragment, task-flow parameters, back-end Java bean, and task-flow activities.

To create the Delete task flow, do the following:

1. Define a task-flow parameter, delHierParamBean, to pass values between the Hierarchy component and the registered Delete task flow in pageFlowScope.

```
<input-parameter-definition>
    <name>delHierParam</name>
    <value>#{pageFlowScope.delHierParam}</value>
    <class>oracle.apps.fnd.applcore.trees.ui.managed.HierParamBean</class>
</input-parameter-definition>
```

2. To capture the event of dismissing the Delete popup window, define Edit task-flow return activities and use them as the ends of the task flow. For example, to define the Submit and Cancel task-flow return activities:

```
<task-flow-return id="Submit">
    <outcome>
        <name>Submit</name>
    </outcome>
</task-flow-return/>

<task-flow-return id="Cancel">
    <outcome>
        <name>Cancel</name>
    </outcome>
</task-flow-return/>
```

Normally, Cancel is a free event. However, you must pass values back when Submit is triggered. Therefore, you can have a submit() actionListener mapping to it.

3. In your back-end bean, using the following code to get the task-flow parameter:

```
import oracle.apps.fnd.appcore.trees.ui.managed.HierParamBean;
...
HierParamBean delParam =
```
For the Delete task flow, you need to know the selected node from the Hierarchy component. In the Delete popup window, delete the node and confirm the deletion. After deleting the node, you must indicate whether or not the deletion was successful. You can perform all of these tasks by calling **HierParamBean** methods.

For example, in the task-flow initializer, you can use the following code to get the primary keys of the selected node:

```java
List<String> pkValue = delParam.getPkValue();
```

In the `submit()` actionListener that maps to the Submit return activity, you specify the result:

```java
public void submit(ActionEvent event)
{
    ...
    delParam.setDeleted(true);
}
```

4. In the page definition file of the page using the Hierarchy component, add the following task-flow entry in the "executables" section:

```xml
<taskFlow id="delTaskflow"
    taskFlowId="#{backingBeanScope.DelNodeBean.delTaskflow}"
    activation="deferred"
    Refresh="ifNeeded"
    xmlns="http://xmlns.oracle.com/adf/controller/binding">
    <parameters>
        <parameter id="delHierParam" xmlns="http://xmlns.oracle.com/adfm/uimodel"
            value="#{pageFlowScope.delHierParam}"/>
    </parameters>
</taskFlow>
```

### 20.9 About Using the fnd:hierarchy Property Inspector to Specify Tree Versions

Using the Hierarchy component to develop applications in JDeveloper requires you to specify a tree structure in the Property Inspector.

To access the Hierarchy-Property Inspector, highlight **fnd:hierarchy** in the Structure window and select the **Hierarchy-Property Inspector** tab.
The Hierarchy component has facets and properties listed in the next tables.

### Table 20-15  Hierarchy Facets

<table>
<thead>
<tr>
<th>Facet</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>hierarchy</td>
<td>Holds ADF Tree or TreeTable</td>
<td>af:tree or af:treeTable</td>
</tr>
<tr>
<td>toolbar</td>
<td>Additional toolbar buttons to be added for custom use</td>
<td>ADF command toolbar buttons under an ADF toolbar</td>
</tr>
</tbody>
</table>

### Table 20-16  Hierarchy Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Unique identification number for hierarchy</td>
<td>string</td>
</tr>
<tr>
<td>rendered</td>
<td>Indicates if the hierarchy is rendered</td>
<td>boolean</td>
</tr>
<tr>
<td>readOnly</td>
<td>Indicates if the hierarchy will render in read-only mode</td>
<td>boolean</td>
</tr>
<tr>
<td>treeStructureCode</td>
<td>Code for the tree structure to be used for the hierarchy</td>
<td>string</td>
</tr>
<tr>
<td>tree Code</td>
<td>Code for the tree to be used for the hierarchy</td>
<td>string</td>
</tr>
<tr>
<td>treeVersionId</td>
<td>ID for the tree version to be used for the hierarchy</td>
<td>string</td>
</tr>
<tr>
<td>actionsMenuRendered</td>
<td>Controls if action menu needs to be rendered</td>
<td>boolean</td>
</tr>
<tr>
<td>toolbarRendered</td>
<td>Controls if toolbar needs to be rendered</td>
<td>boolean</td>
</tr>
<tr>
<td>addVisible</td>
<td>Controls if add action is visible</td>
<td>boolean</td>
</tr>
<tr>
<td>addRendered</td>
<td>Controls if add action is rendered</td>
<td>boolean</td>
</tr>
<tr>
<td>addDisabled</td>
<td>Controls if add action is disabled</td>
<td>boolean</td>
</tr>
</tbody>
</table>
Table 20-16  (Cont.) Hierarchy Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>addText</td>
<td>Custom text to be used for add action</td>
<td>string</td>
</tr>
<tr>
<td>createVisible</td>
<td>Controls if create action is visible</td>
<td>boolean</td>
</tr>
<tr>
<td>createRendered</td>
<td>Controls if create action is rendered</td>
<td>boolean</td>
</tr>
<tr>
<td>createDisabled</td>
<td>Controls if create action is disabled</td>
<td>boolean</td>
</tr>
<tr>
<td>createText</td>
<td>Custom text to be used for create action</td>
<td>string</td>
</tr>
<tr>
<td>duplicateVisible</td>
<td>Controls if duplicate action is visible</td>
<td>boolean</td>
</tr>
<tr>
<td>duplicateRendered</td>
<td>Controls if duplicate action is rendered</td>
<td>boolean</td>
</tr>
<tr>
<td>duplicateDisabled</td>
<td>Controls if duplicate action is disabled</td>
<td>boolean</td>
</tr>
<tr>
<td>duplicateText</td>
<td>Custom text to be used for duplicate action</td>
<td>string</td>
</tr>
<tr>
<td>editVisible</td>
<td>Controls if edit action is visible</td>
<td>boolean</td>
</tr>
<tr>
<td>editRendered</td>
<td>Controls if edit action is rendered</td>
<td>boolean</td>
</tr>
<tr>
<td>editDisabled</td>
<td>Controls if edit action is disabled</td>
<td>boolean</td>
</tr>
<tr>
<td>editText</td>
<td>Custom text to be used for edit action</td>
<td>string</td>
</tr>
<tr>
<td>removeVisible</td>
<td>Controls if remove action is visible</td>
<td>boolean</td>
</tr>
<tr>
<td>removeRendered</td>
<td>Controls if remove action is rendered</td>
<td>boolean</td>
</tr>
<tr>
<td>removeDisabled</td>
<td>Controls if remove action is disabled</td>
<td>boolean</td>
</tr>
<tr>
<td>removeText</td>
<td>Custom text to be used for remove action</td>
<td>string</td>
</tr>
<tr>
<td>deleteVisible</td>
<td>Controls if delete action is visible</td>
<td>boolean</td>
</tr>
<tr>
<td>deleteRendered</td>
<td>Controls if delete action is rendered</td>
<td>boolean</td>
</tr>
<tr>
<td>deleteDisabled</td>
<td>Controls if delete action is disabled</td>
<td>boolean</td>
</tr>
<tr>
<td>deleteText</td>
<td>Custom text to be used for delete action</td>
<td>string</td>
</tr>
<tr>
<td>registerTaskflow</td>
<td>Specifies whether to use task flow (true) or .jspx file (false). For example:</td>
<td>boolean</td>
</tr>
</tbody>
</table>

```xml
<dnd:hierarchy registerTaskflow="true">
...  
</dnd:hierarchy>
```

**Note:**

Since only customers create tree versions, you must use service APIs to generate lists of tree versions or active tree versions.
The data that displays in your application depends on the tree structure you specify in the Property Inspector. The tree structure automatically determines the following at run time:

- The tree available under this particular tree structure. If there are multiple trees, the first one is chosen.
- The active tree version (for the current date) available under the tree. If there are multiple tree versions, the first one is chosen.

20.10 About Using the Expression Builder to Bind TreeCode, TreeStructureCode, and TreeVersionId Properties

You also can use the Expression Builder to bind some of the properties mentioned in the previous section. They are the following:

- TreeCode
- TreeStructureCode
- TreeVersionId

Use the following expressions:

**TreeCode expression:**

${HierarchyHandler.treeModelsList[hierarchyId].treeCode}$

**TreeStructureCode expression:**

${HierarchyHandler.treeModelsList[hierarchyId].treeStructureCode}$

**TreeVersionId expression:**

${HierarchyHandler.treeModelsList[hierarchyId].treeVersionId}$

---

**Note:**

The hierarchyId variable is the ID of the Hierarchy component.

---

20.11 Embed the Tree Picker Component in a User Interface

Tree Picker is a reusable Oracle ADF task flow, similar to a date picker, that enables you to select tree data from a list of values. It is found in the Trees-View.jar section of the Component Palette.

To add a Tree Picker component to your user interface, do the following:

1. On the page from which the Tree Picker will be launched, create an icon or button with Action set to launch.
2. Drag and drop the Tree Picker from Component Palette > Trees-View.jar onto your task flow.
3. Connect the launch page to the Tree Picker using an appropriate Control Flow Case with from-outcome set to launch and run-as-dialog set to true.
4. In the **Property Inspector > Parameters** section for the task flow, enter the appropriate parameters for the following:

- **treeStructureCode** (required) - The code assigned to the tree structure. For example, FND_DEMO_EMP_TS.
- **treeCode** (optional) - The code assigned to the tree. For example, FND_DEMO_EMP_T.
- **treeVersionId** (optional) - String representation of an automatically generated binary value (Raw) identifier. You cannot hard code a TreeVersionId.
- **selectMode** (optional) - A parameter to control the row selection behavior of tree table on a user interface page. Acceptable values are single and multiple. The default is single.

The Tree Picker returns a list of TreeNodes:

- If the Tree Picker is launched as a popup window, the return value can be obtained in the returnListener of the icon or button using:
  
  ```java
  List<TreeNode> selectedTreeNodes = List<TreeNode>()event.getReturnValue();
  ```

- If the Tree Picker is launched in place, the return value can be obtained from the pageFlowScope using:
  
  ```java
  Map pageFlow = AdfFacesContext.getCurrentInstance().getPageFlowScope();List <TreeNode> selectedNodes = (List<TreeNode>()) pageFlow.get("return TreeNodes");
  ```

The following picture shows an example of a Tree Picker user interface.

**Figure 20-78  Example Tree Picker**

The following picture shows the results window that displays when you enter a tree-structure-code value and click **Select Tree Node**.
20.12 About Setting Bind Variables and View Criteria

Every data source requires a view object. If the data source view object has bind variables and view criteria that tree management needs to apply, you must set them manually in JDeveloper.

20.12.1 How to Set Bind Variables and View Criteria

Use the following procedure to set bind variables and view criteria.

1. In JDeveloper, click on the appropriate data source view object. (Ensure that it is highlighted in the Structure window.)

2. Select the Query option from the Overview tab. The following picture shows the Bind Variables and View Criteria sections after the Query option is selected.
3. Do any of the following:
   - Click the **Bind Variables Add** icon to add any necessary bind variables.
   - Click the **View Criteria Add** icon to add any necessary view criteria.

### 20.13 About Using Service APIs to Manage Trees

An application programming interface, or **API**, is a source code interface that a library provides to support requests for services to be made of it by computer programs. In other words, APIs provide the building blocks that make it easier to develop these programs. Although an API specifies an interface and the behavior of the identifiers specified in that interface, it does not specify how the behavior might be implemented.

There are three service application modules that you can use to interact with the tree management infrastructure:

- **TreeStructureService**
- **TreeService**
- **TreeNodeService**

**Note:**

No Service Data Objects (SDOs) are provided and these application modules must be instantiated and invoked in a co-located mode.
20.13.1 How to Use TreeStructureService

The TreeStructureService application module is defined in oracle.apps.fnd.applicore.trees.service.applicationModule.TreeStructureService and allows access to tree structure metadata. This application module exposes the TreeStructureVO under the name “TreeStructure” as well as the hierarchy of ADF Business Components objects accessible through the TreeStructureVO. This application module does not include any of the tree or tree version entities. The Javadoc for the available APIs is included with JDeveloper. To access the Javadoc in JDeveloper, do the following:

1. Choose the Go to Java Class... option from the Navigate dropdown menu.
   
The Go to Java Class window opens.

2. Enter TreeStructureService in the Name: field.

3. Choose Go to: > Javadoc and click OK.

This application module is considered a public API to work with tree structure metadata and exposes the APIs shown in the following table.

<table>
<thead>
<tr>
<th>Table 20-17 TreeStructureService APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
</tr>
<tr>
<td>--------------------------</td>
</tr>
<tr>
<td>getTreeStructure</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>getRootDataSourceRels</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>getAllTreeColumns</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>getAllDataSources</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>getTreeNodeTable</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>duplicateTreeStructure</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
### Table 20-17 (Cont.) TreeStructureService APIs

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteTreeStructure</td>
<td>void deleteTreeStructure(String treeStructureCode)</td>
</tr>
<tr>
<td></td>
<td>This API is a Java front end to the PL/SQL API FND_TREE_UTILS.delete_tree_structure and is used to delete a tree structure. This deletes all underlying trees and tree versions associated with this tree structure (including flattened data, if any).</td>
</tr>
</tbody>
</table>

#### 20.13.2 How to Use TreeService

The TreeService application module is defined in `oracle.apps.fnd.applcore.trees.service.applicationModule.TreeService` and provides access to trees and tree versions. TreeService also provides flattening APIs. The Javadoc for the available APIs is included with JDeveloper.

To access the Javadoc, do the following:

1. Choose the Go to Java Class... option from the Navigate dropdown menu.
   
The Go to Java Class window opens.

2. Enter TreeService in the Name: field.

3. Choose Go to: > Javadoc and click OK.

This application module is considered a public API to work with trees and tree versions and exposes the APIs shown in the following table.

### Table 20-18 TreeService APIs

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
<th>Optional Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>getTreeRows</td>
<td>RowIterator getTreeRows(String treeStructureCode);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This API returns all trees associated with a given tree structure.</td>
<td></td>
</tr>
<tr>
<td>getTreeCodes</td>
<td>List&lt;String&gt; getTreeCodes(String treeStructureCode);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This API returns a list of tree codes associated with a given tree structure.</td>
<td></td>
</tr>
<tr>
<td>findTree</td>
<td>FndTreeVORow findTree(String treeStructureCode, String treeCode);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This API is used to find a specific tree given its tree structure code and tree code.</td>
<td></td>
</tr>
<tr>
<td>createTree</td>
<td>void createTree(String treeStructureCode, String treeCode, String treeDescription);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This API is used to create a tree for a given tree structure.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>void createTree(String treeStructureCode, String treeCode, String treeName, String treeDescription);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This API is used to create a tree for a given tree structure.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 20-18  (Cont.) TreeService APIs

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
<th>Optional Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>updateTree</td>
<td>void updateTree(String treeStructureCode, String treeCode, String updatedTreeName, String updatedTreeDescription);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This API is used to update details of a tree for a given tree structure.</td>
<td></td>
</tr>
<tr>
<td>updateTreeDataSourceParamValues</td>
<td>void updateTreeDataSourceParamValues(String treeStructureCode, String treeCode, String dataSourceId, String paramId, String paramValue);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This API is used to update the value of the data source parameters for a given tree. The parameter values customized at the tree level will override the default values specified at the tree-structure level.</td>
<td></td>
</tr>
<tr>
<td>duplicateTree</td>
<td>void duplicateTree(String treeStructureCode, String treeCode, String duplicateTreeCode);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This API duplicates a specific tree and assigns a specified tree code to the duplicate. It is a front end to the FND_TREE_UTILS.duplicate_tree PL/SQL API.</td>
<td></td>
</tr>
<tr>
<td>deleteTree</td>
<td>void deleteTree(String treeStructureCode, String treeCode);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This API deletes a tree, all its associated tree versions, including flattened data. It is a front end to the FND_TREE_UTILS.delete_tree PL/SQL API.</td>
<td></td>
</tr>
<tr>
<td>getAllTreeVersions</td>
<td>List&lt;String&gt; getAllTreeVersions(String treeStructureCode, String treeCode);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This API returns a list of all tree versions associated with a given tree.</td>
<td></td>
</tr>
<tr>
<td>getTreeVersions</td>
<td>List&lt;String&gt; getTreeVersions(String treeStructureCode, String treeCode, Timestamp asOfDate);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This API returns a list of tree versions associated with a given tree as of a particular date.</td>
<td></td>
</tr>
<tr>
<td>getCurrentTreeVersions</td>
<td>List&lt;String&gt; getCurrentTreeVersions(String treeStructureCode, String treeCode);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This API returns a list of tree versions associated with a given tree as of the current date.</td>
<td></td>
</tr>
<tr>
<td>findTreeVersion</td>
<td>FndTreeVersionVORow findTreeVersion(String treeStructureCode, String treeCode, String treeVersionId);</td>
<td></td>
</tr>
<tr>
<td></td>
<td>This API is used to locate a specific tree version given its tree structure code, tree code and tree version ID.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 20-18 (Cont.) TreeService APIs

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
<th>Optional Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>createTreeVersion</td>
<td>String createTreeVersion(String treeStructureCode, String treeCode, String treeVersionName, String effectiveStartDate); This API creates a tree version and returns the ID of the created tree version. All parameters are mandatory.</td>
<td>treeVersionDescription, effectiveEndDate, treeVersionNote</td>
</tr>
<tr>
<td></td>
<td>String createTreeVersion(String treeStructureCode, String treeCode, String treeVersionName, String treeVersionDescription, String effectiveStartDate, String effectiveEndDate, String treeVersionNote); This API creates a tree version and returns the ID of the created tree version. The default values for all these parameters is NULL. The default for the effectiveEndDate parameter is 31-DEC-4712.</td>
<td></td>
</tr>
<tr>
<td>updateTreeVersion</td>
<td>String updateTreeVersion(String treeStructureCode, String treeCode, String treeVersionName, String updatedTreeVersionName, String updatedTreeVersionDescription, String updatedEffectiveStartDate, String updatedEffectiveEndDate, String updatedTreeVersionNote); This API updates the specified tree version and returns the ID of the updated tree version for a given tree structure and tree code.</td>
<td>effectiveEndDate updatedTreeVersionName, updatedTreeVersionDescription, updatedEffectiveStartDate, updatedEffectiveEndDate, updatedTreeVersionNote</td>
</tr>
<tr>
<td>duplicateTreeVersion</td>
<td>String duplicateTreeVersion(String treeStructureCode, String treeCode, String treeVersionId, String treeVersionName); This API is a front end to the PL/SQL API FND_TREE_UTILS.duplicate_tree_version and duplicates a specific tree version. The API returns the auto-generated ID of the duplicate tree version.</td>
<td></td>
</tr>
<tr>
<td>deleteTreeVersion</td>
<td>void deleteTreeVersion(String treeStructureCode, String treeCode, String treeVersionId); This API is a front end to the PL/SQL API FND_TREE_UTILS.delete_tree_version and deletes a tree version including its flattened data (if any).</td>
<td></td>
</tr>
<tr>
<td>rowFlatten</td>
<td>void rowFlatten(String treeStructureCode, String treeCode, String treeVersionId); This API row-flattens a specific tree version.</td>
<td></td>
</tr>
<tr>
<td>columnFlatten</td>
<td>void columnFlatten(String treeStructureCode, String treeCode, String treeVersionId); This API column-flattens a specific tree version.</td>
<td></td>
</tr>
</tbody>
</table>

Organize Hierarchical Data with Tree Structures 20-91
20.13.3 How to Use TreeNodeService

The TreeNodeService application module is defined in oracle.apps.fnd.applcore.trees.service.applicationModule.TreeNodeService and provides the core node operations such as adding and deleting nodes. The APIs support three types of nodes:

- value nodes
- range nodes
- tree-in-tree nodes.

The Java APIs are covers to the PL/SQL APIs that are provided in the FND_TREE_UTILS PL/SQL package.

To access the Javadoc, do the following:

1. Choose the Go to Java Class... option from the Navigate dropdown menu.
   
   The Go to Java Class window opens.

2. Enter TreeNodeService in the Name: field.

3. Choose Go to: > Javadoc and click OK.

This application module is considered a public API to work with tree nodes and exposes the APIs shown in the next table.

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addValueTreeNode</td>
<td>String addValueTreeNode(String treeStructureCode, String treeCode, String treeVersionId, String parentTreeNodeId, String dataSourceId, String pk1Value, String pk2Value, String pk3Value, String pk4Value, String pk5Value);</td>
</tr>
<tr>
<td></td>
<td>String addValueTreeNode(String treeStructureCode, String treeCode, String treeVersionId, String parentTreeNodeId, String dataSourceId, String pk1Value, String pk2Value, String pk3Value, String pk4Value, String pk5Value, String treeLabelId);</td>
</tr>
<tr>
<td></td>
<td>This API adds a value-based tree node to a specific tree version. It is a front end to the PL/SQL API FND_TREE_UTILS.add_value_tree_node. It returns the tree node ID of the newly added node. The API has two signatures - one that takes in a tree label to be associated with the tree node and one that does not.</td>
</tr>
</tbody>
</table>
### Table 20-19 (Cont.) TreeNodeService APIs

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addRangeTreeNode</td>
<td>String addRangeTreeNode(String treeStructureCode, String treeCode, String treeVersionId, String parentTreeNodeId, String dataSourceId, String pk1StartValue, String pk2StartValue, String pk3StartValue, String pk4StartValue, String pk5StartValue, String pk1EndValue, String pk2EndValue, String pk3EndValue, String pk4EndValue, String pk5EndValue); String addRangeTreeNode(String treeStructureCode, String treeCode, String treeVersionId, String parentTreeNodeId, String dataSourceId, String pk1StartValue, String pk2StartValue, String pk3StartValue, String pk4StartValue, String pk5StartValue, String pk1EndValue, String pk2EndValue, String pk3EndValue, String pk4EndValue, String pk5EndValue, String treeLabelId); This API adds a range-based tree node to a specific tree version. It is a front end to the PL/SQL API FND_TREE_UTILS.add_range_tree_node. It returns the tree node ID of the newly added node. The API has two signatures - one that takes in a tree label to be associated with the tree node and one that does not.</td>
</tr>
<tr>
<td>addTreeTreeNode</td>
<td>String addTreeTreeNode(String treeStructureCode, String treeCode, String treeVersionId, String parentTreeNodeId, String referenceTreeCode, String referenceTreeVersionId); String addTreeTreeNode(String treeStructureCode, String treeCode, String treeVersionId, String parentTreeNodeId, String referenceTreeCode, String referenceTreeVersionId, String treeLabelId); This API adds a tree node that references another tree version. It is a front end to the PL/SQL API FND_TREE_UTILS.add_tree_tree_node. It returns the tree node ID of the newly added node. The API has two signatures - one that takes in a tree label to be associated with the tree node and one that does not.</td>
</tr>
<tr>
<td>deleteTreeNode</td>
<td>void deleteTreeNode(String treeStructureCode, String treeCode, String treeVersionId, String treeNodeId); This API is a front end to the PL/SQL API FND_TREE_UTILS.delete_tree_node and deletes a specific tree node. Any children of that node are automatically promoted up the hierarchy.</td>
</tr>
<tr>
<td>updateTreeNode</td>
<td>void updateTreeNode(String treeStructureCode, String treeCode, String treeVersionId, String parentTreeNodeId, String dataSourceId, String pk1StartValue, String pk2StartValue, String pk3StartValue, String pk4StartValue, String pk5StartValue, String pk1EndValue, String pk2EndValue, String pk3EndValue, String pk4EndValue, String pk5EndValue, String referenceTreeCode, String referenceTreeVersionId, String treeLabelId); This API is a front end to the PL/SQL API FND_TREE_UTILS.update_tree_node and is used to update the data associated with a specific tree node. It cannot be used to move the tree node.</td>
</tr>
</tbody>
</table>
Table 20-19  (Cont.) TreeNodeService APIs

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>moveTreeNode</td>
<td>void moveTreeNode(String treeStructureCode, String treeCode, String treeVersionId, String treeNodeId, String destinationParentNodeId);</td>
</tr>
<tr>
<td></td>
<td>This API is used to move a tree node within the hierarchy. The entire sub-tree rooted at the node being moved is moved. This API is a front end to the PL/SQL API FND_TREE_UTILS.move_tree_node.</td>
</tr>
<tr>
<td>moveSubTree</td>
<td>String moveSubTree(String treeStructureCode, String treeCode, String treeVersionName, String treeNodeId, String destParentTreeNodeId, String destParentTreeVersionName, String destParentTreeCode, String newTreeNodeId);</td>
</tr>
<tr>
<td></td>
<td>This API moves a sub-tree under the specified parent tree node within the same tree or across trees and returns the ID of the tree node that has been moved.</td>
</tr>
<tr>
<td>copySubTree</td>
<td>String copySubTree(String treeStructureCode, String treeCode, String treeVersionName, String treeNodeId, String destParentTreeNodeId, String destParentTreeVersionName, String destParentTreeCode, String newTreeNodeId);</td>
</tr>
<tr>
<td></td>
<td>This API copies a sub tree under specified parent Tree Node within the same tree or across trees and returns the ID of the tree node that has been copied.</td>
</tr>
<tr>
<td>removeSubTree</td>
<td>void removeSubTree(String treeStructureCode, String treeCode, String treeVersionName, String treeNodeId);</td>
</tr>
<tr>
<td></td>
<td>This API removes sub tree under the specified parent tree node.</td>
</tr>
<tr>
<td>findValueTreeNodes</td>
<td>RowIterator findValueTreeNodes(String treeStructureCode, String treeCode, String treeVersionId, String[] pkValues);</td>
</tr>
<tr>
<td></td>
<td>This API is used to find all value tree nodes with the specified primary key.</td>
</tr>
<tr>
<td>findRangeTreeNodes</td>
<td>RowIterator findRangeTreeNodes(String treeStructureCode, String treeCode, String treeVersionId, String[] pkStartValues, String[] pkEndValues);</td>
</tr>
<tr>
<td></td>
<td>This API is used to find all range tree nodes with the specified range.</td>
</tr>
<tr>
<td>findRefTreeNodes</td>
<td>RowIterator findRefTreeNodes(String treeStructureCode, String treeCode, String treeVersionId, String refTreeCode, String refTreeVersionId);</td>
</tr>
<tr>
<td></td>
<td>This API is used to find all tree nodes that reference the specified tree tree version.</td>
</tr>
</tbody>
</table>

20.14 Advanced Topics

This section includes information about the following advanced topics:

- Tree data model
- PL/SQL APIs
- Web services APIs
- Incremental flattening
• Oracle Enterprise Scheduler Service for auditing and flattening trees
• Trees business events
• WebLogic Scripting Tool (WLST) commands for flattening
• XML report formats for WLST commands
• Purging unused tree logs

20.14.1 About Using the Tree Data Model

The following are new or modified tables and views that are used by and relevant to the Tree Management infrastructure. They are set up in the FUSION schema.

Tables:

• FND_TREE_STRUCTURE
• FND_TREE_STRUCTURE_TL
• FND_TS_DATA_SOURCE
• FND_TS_DATA_SOURCE_REL
• FND_TS_DATA_SOURCE_PARAMS
• FND_LABEL
• FND_LABEL_TL
• FND_TREE
• FND_TREE_TL
• FND_TREE_DATA_SOURCE_PARAMS
• FND_TREE_VERSION
• FND_TREE_VERSION_TL
• FND_NODE
• FND_NODE_TL
• FND_TREE_LABEL
• FND_TREE_NODE
• FND_TREE_NODE_RF
• FND_TREE_NODE_CF
• FND_TREE_AUDIT_JOB
• FND_TREE_VERSION_AUDIT_RES
• FND_TREE_VERSION_AUDIT_RES_TL
• FND_TREE_LOG
• FND_TREE_LOG_PARAMS
20.14.2 About Using PL/SQL APIs

Tree Management provides public PL/SQL APIs to work with trees. You can find these APIs in the PL/SQL package FND_TREE_UTILS in the FUSION schema.

**Note:**
The PL/SQL package FND_TREE_UTILS_PVT contains private APIs for internal use with Oracle Fusion tree management. No other use of these APIs is supported.

20.14.3 About Using Web Services APIs

Oracle Fusion Applications uses the trees framework for modeling several hierarchies, and for managing tree structures, trees, and tree versions. In addition to APIs, hierarchical components, and task flows, the framework exposes web services to accomplish integration.

The following tables include web services APIs for TreeService and TreeNodeService.

### Table 20-20  Web Services APIs for TreeService

<table>
<thead>
<tr>
<th>Operation</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create tree</td>
<td>public void createTree(String treeStructureCode, String treeCode, String treeName);</td>
</tr>
<tr>
<td></td>
<td>Creates a tree.</td>
</tr>
<tr>
<td>Update tree</td>
<td>public void updateTree(String treeStructureCode, String treeCode, String updatedTreeName, String updatedTreeDescription);</td>
</tr>
<tr>
<td></td>
<td>Updates a tree.</td>
</tr>
<tr>
<td>Update tree data source parameter values</td>
<td>public void updateTreeDataSourceParamValues(String treeStructureCode, String treeCode, String dataSourceId, String paramId, String paramValue);</td>
</tr>
<tr>
<td></td>
<td>Updates the value of the data source parameter for a given tree. Parameter values customized at the tree level will override the default values specified at the tree-structure level.</td>
</tr>
</tbody>
</table>
### Table 20-20  *(Cont.)* Web Services APIs for TreeService

<table>
<thead>
<tr>
<th>Operation</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duplicate tree</td>
<td>public void duplicateTree(String treeStructureCode, String treeCode, String duplicateTreeCode); Duplicates a tree without duplicating the underlying tree versions.</td>
</tr>
<tr>
<td>Delete tree</td>
<td>public void deleteTree(String treeStructureCode, String treeCode); Deletes a tree.</td>
</tr>
<tr>
<td>Create tree version</td>
<td>public String createTreeVersion(String treeStructureCode, String treeCode, String treeVersionName, String treeVersionDescription, String effectiveStartDate, String effectiveEndDate, String treeVersionNote); Creates a tree version and returns a string with the ID of the tree version that has been created.</td>
</tr>
<tr>
<td>Update tree version</td>
<td>public String updateTreeVersion(String treeStructureCode, String treeCode, String treeVersionName, String updatedTreeVersionName, String updatedTreeVersionDescription, String updatedEffectiveStartDate, String updatedEffectiveEndDate, String updatedTreeVersionNote); Updates a tree version.</td>
</tr>
<tr>
<td>Duplicate tree version</td>
<td>public String duplicateTreeVersion(String treeStructureCode, String treeCode, String treeVersionName, String duplicateTreeVersionName); Duplicates a tree version.</td>
</tr>
<tr>
<td>Delete tree version</td>
<td>public void deleteTreeVersion(String treeStructureCode, String treeCode, String treeVersionName); Deletes the specified tree version.</td>
</tr>
<tr>
<td>Row flatten tree version</td>
<td>public void rowFlatten(String treeStructureCode, String treeCode, String treeVersionName); Row flattens a tree version.</td>
</tr>
<tr>
<td>Column flatten tree version</td>
<td>public void columnFlatten(String treeStructureCode, String treeCode, String treeVersionName); Column flattens a tree version.</td>
</tr>
</tbody>
</table>

### Table 20-21  Web Services APIs for TreeNodeService

<table>
<thead>
<tr>
<th>Operation</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add tree value node</td>
<td>public String addRangeTreeNode(String treeStructureCode, String treeVersionName, String parentTreeNodeId, String dataSourceId, String pk1StartValue, String pk2StartValue, String pk3StartValue, String pk4StartValue, String pk5StartValue, String pk1EndValue, String pk2EndValue, String pk3EndValue, String pk4EndValue, String pk5EndValue, String treeLabelId); Adds a value-based tree node and returns the ID of the tree node that was added.</td>
</tr>
</tbody>
</table>
### Table 20-21 (Cont.) Web Services APIs for TreeNodeService

<table>
<thead>
<tr>
<th>Operation</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add tree range node</td>
<td>public void updateTree(String treeStructureCode, String treeCode, String updatedTreeNodeName, String updatedTreeDescription);</td>
</tr>
<tr>
<td></td>
<td>Adds a range-based tree node and returns the ID of the tree node that was added.</td>
</tr>
<tr>
<td>Add reference tree node</td>
<td>public String addRangeTreeNode(String treeStructureCode, String treeCode, String treeVersionName, String parentTreeNodeId, String dataSourceId, String pk1StartValue, String pk2StartValue, String pk3StartValue, String pk4StartValue, String pk5StartValue, String pk1EndValue, String pk2EndValue, String pk3EndValue, String pk4EndValue, String pk5EndValue, String treeLabelId);</td>
</tr>
<tr>
<td></td>
<td>Adds a reference tree node and returns the ID of the tree node that was added.</td>
</tr>
<tr>
<td>Delete tree node</td>
<td>public void deleteTreeNode(String treeStructureCode, String treeCode, String treeVersionName, String treeNodeId);</td>
</tr>
<tr>
<td></td>
<td>Deletes a tree node.</td>
</tr>
<tr>
<td>Move tree node</td>
<td>public void moveTreeNode(String treeStructureCode, String treeCode, String treeVersionName, String treeNodeId, String destParentTreeNodeId);</td>
</tr>
<tr>
<td></td>
<td>Moves a tree node under specified parent tree node.</td>
</tr>
<tr>
<td>Update tree node</td>
<td>public void updateTreeNode(String treeStructureCode, String treeCode, String treeVersionName, String treeNodeId, String parentTreeNodeId, String dataSourceId, String pk1StartValue, String pk2StartValue, String pk3StartValue, String pk4StartValue, String pk5StartValue, String pk1EndValue, String pk2EndValue, String pk3EndValue, String pk4EndValue, String pk5EndValue, String treeLabelId);</td>
</tr>
<tr>
<td></td>
<td>Updates an existing tree node.</td>
</tr>
<tr>
<td>Move sub-tree</td>
<td>public String moveSubTree(String treeStructureCode, String treeCode, String treeVersionName, String treeNodeId, String destParentTreeNodeId, String destParentTreeVersionName, String destParentTreeCode, String newTreeNodeId);</td>
</tr>
<tr>
<td></td>
<td>Moves a sub-tree under specified parent tree node within the same tree or across trees.</td>
</tr>
<tr>
<td>Copy sub-tree</td>
<td>public String copySubTree(String treeStructureCode, String treeCode, String treeVersionName, String treeNodeId, String destParentTreeNodeId, String destParentTreeVersionName, String destParentTreeCode, String newTreeNodeId);</td>
</tr>
<tr>
<td></td>
<td>Copies a sub-tree under specified parent tree node within the same tree or across trees.</td>
</tr>
<tr>
<td>Remove sub-tree</td>
<td>public void removeSubTree(String treeStructureCode, String treeCode, String treeVersionName, String treeNodeId);</td>
</tr>
<tr>
<td></td>
<td>Removes the sub-tree under the specified parent tree node.</td>
</tr>
</tbody>
</table>
20.14.4 About Using Incremental Flattening

Incremental flattening optimizes the process by starting with the results of a previous flattening instead of flattening the data from scratch.

Figure 20-81  Flattening Delta

To flatten incrementally, a delta of flattening operations that occurred between these two sets of start and end points is created, and information about what happened during those operations is stored in three tables:

- FND_TREE_FLATTENING_HISTORY
- FND_TREE_LOG
- FND_TREE_LOG_PARAMS

These tables are described in the sections that follow.

20.14.4.1 About Using FND_TREE_FLATTENING_HISTORY

This table records the flattening history for a specific tree version. For optimization, only the last process point is recorded. Process_Point records the time of the last tree-node operation that has been flattened.

The following table shows the contents of the FND_TREE_FLATTENING_HISTORY table.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Nullable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree_Structure_Code (Primary Key)</td>
<td>Varchar2(30)</td>
<td>No</td>
</tr>
<tr>
<td>Tree_Code (Primary Key)</td>
<td>Varchar2(30)</td>
<td>No</td>
</tr>
<tr>
<td>Tree_Version_ID (Primary Key)</td>
<td>Varchar2(32)</td>
<td>No</td>
</tr>
<tr>
<td>Process_Point</td>
<td>Timestamp(6)</td>
<td>No</td>
</tr>
<tr>
<td>Flattening_Type (Primary Key)</td>
<td>Varchar2(32)</td>
<td>No</td>
</tr>
<tr>
<td>Created_By</td>
<td>Varchar2(64)</td>
<td>No</td>
</tr>
<tr>
<td>Creation_Date</td>
<td>Timestamp(6)</td>
<td>No</td>
</tr>
<tr>
<td>Last_Updated_By</td>
<td>Varchar2(64)</td>
<td>No</td>
</tr>
<tr>
<td>Last_Update_Date</td>
<td>Timestamp(6)</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 20-22  (Cont.) FND_TREE_FLATTENING_HISTORY

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Nullable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last_Update_Login</td>
<td>Varchar2(32)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

20.14.4.2 About Using FND_TREE_LOG

FND_TREE_LOG is a log of all flattening operations for one tree version. The log enables database administrators (DBAs) to move data easily and efficiently to external systems such as a data warehouse, or from test to production systems.

FND_TREE_LOG stores tree-node operations. For each specific tree-version operation, a unique Log_Entry_ID is assigned and the operation type is logged.

There are three types of tree nodes:

- value
- range
- tree node

There also are three types of tree-node operations:

- add
- move
- delete

Subsequently, there are nine types of operations:

- add value node
- move value node
- delete value node
- add range node
- move range node
- delete range node
- add tree node
- move tree node
- delete tree node

The following table shows the contents of the FND_TREE_LOG table.

Table 20-23  FND_TREE_LOG

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Nullable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log_Entry_ID (Primary Key)</td>
<td>Varchar2(32)</td>
<td>No</td>
</tr>
<tr>
<td>Tree_Structure_Code</td>
<td>Varchar2(30)</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 20-23  (Cont.) FND_TREE_LOG

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Nullable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree_Code</td>
<td>Varchar2(30)</td>
<td>No</td>
</tr>
<tr>
<td>Tree_Version_ID</td>
<td>Varchar2(32)</td>
<td>No</td>
</tr>
<tr>
<td>Operation_Type</td>
<td>Varchar1(32)</td>
<td>No</td>
</tr>
<tr>
<td>Created_By</td>
<td>Varchar2(64)</td>
<td>No</td>
</tr>
<tr>
<td>Creation_Date</td>
<td>Timestamp(6)</td>
<td>No</td>
</tr>
<tr>
<td>Last_Updated_By</td>
<td>Varchar2(64)</td>
<td>No</td>
</tr>
<tr>
<td>Last_Update_Date</td>
<td>Timestamp(6)</td>
<td>No</td>
</tr>
<tr>
<td>Last_Update_Login</td>
<td>Varchar2(32)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

20.14.4.3 About Using FND_TREE_LOG_PARAMS

Since the FND_TREE_LOG table does not record parameters for each operation, the FND_TREE_LOG_PARAMS table is used to log them. The two tables are referenced by a foreign key, Log_Entry_ID. This design helps save space and clearly organizes the information.

The following table shows the contents of the FND_TREE_LOG_PARAMS table.

Table 20-24  FND_TREE_LOG_PARAMS

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Nullable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log_Entry_ID (PK)</td>
<td>Varchar2(32)</td>
<td>No</td>
</tr>
<tr>
<td>Param_Name (PK)</td>
<td>Varchar2(64)</td>
<td>No</td>
</tr>
<tr>
<td>Param_Value</td>
<td>Varchar2(100)</td>
<td>No</td>
</tr>
<tr>
<td>Created_By</td>
<td>Varchar2(64)</td>
<td>No</td>
</tr>
<tr>
<td>Creation_Date</td>
<td>Timestamp(6)</td>
<td>No</td>
</tr>
<tr>
<td>Last_Updated_By</td>
<td>Varchar2(64)</td>
<td>No</td>
</tr>
<tr>
<td>Last_Update_Date</td>
<td>Timestamp(6)</td>
<td>No</td>
</tr>
<tr>
<td>Last_Update_Login</td>
<td>Varchar2(32)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

20.14.4.4 About Row Flattening

Row-flattening results are stored in the table registered as the row-flattening table for the tree structure. If you register a custom row-flattening table for your tree structure, ensure it has the same schema as FND_TREE_NODE_RF.

IS_LEAF and DISTANCE are two important row-flattening-table columns. For more information, see the following sections.
20.14.4.1 About IS_LEAF

This column provides information about whether or not a tree node is a leaf. In many instances, only a leaf contain meaningful information, while other nodes provide a structural purpose. IS_LEAF makes it easier to differentiate a leaf from other nodes, and makes it simpler to write simple queries and get faster responses. Valid values are Y (yes) and N (no).

20.14.4.2 About DISTANCE

This column indicates the distance between the node and its ancestor, which is specified in the row. For example, the distance between a node and its parent or children is 1, between a node and its grandparent or grandchildren is 2, and so on. DISTANCE, then, helps developers to get the entire path - from the root node to the intermediate leaf/node without having to perform any additional queries.

In the FND_TREE_NODE table, DISTANCE is stored in the form of an adjacency list, as shown in the next table.

![Figure 20-82 Example of DISTANCE](image)

Table 20-25 Adjacency List

<table>
<thead>
<tr>
<th>Node</th>
<th>Parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Null</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>C</td>
</tr>
</tbody>
</table>

After flattening, DISTANCE is stored in the FND_TREE_NODE table.

---

**Note:**

The node ancestor also includes itself.

---

Table 20-26 Flattened FND_TREE_NODE Table

<table>
<thead>
<tr>
<th>Node</th>
<th>Ancestor</th>
<th>Distance</th>
<th>IS_LEAF?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Null</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>0</td>
<td>N</td>
</tr>
<tr>
<td>B</td>
<td>Null</td>
<td>2</td>
<td>Y</td>
</tr>
</tbody>
</table>
### Table 20-26  (Cont.) Flattened FND_TREE_NODE Table

<table>
<thead>
<tr>
<th>Node</th>
<th>Ancestor</th>
<th>Distance</th>
<th>IS_LEAF?</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>A</td>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>0</td>
<td>Y</td>
</tr>
<tr>
<td>C</td>
<td>Null</td>
<td>2</td>
<td>N</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>1</td>
<td>N</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>0</td>
<td>N</td>
</tr>
<tr>
<td>D</td>
<td>Null</td>
<td>3</td>
<td>Y</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>2</td>
<td>Y</td>
</tr>
<tr>
<td>D</td>
<td>C</td>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>0</td>
<td>Y</td>
</tr>
</tbody>
</table>

To find the path from the root of D, the query would be the following:

```sql
select * 
from fnd_tree_node_rf 
where tree_node_id = D order by distance
```

### 20.14.4.5 About Column Flattening

Column flattening generally applies only to level-based trees. In the case of a view object for Business Intelligence (BICVO), however, value-based trees also can be column-flattened. (For more information, see Designing and Securing View Objects for.)

For level-based trees, the maximum level of a tree version is 32. Subsequently, if a tree version is not level-based or if it has a tree-version level that exceeds 32, only row flattening can be performed.

Column-flattening results are stored in the table specified in the tree structure, which has the same schema as FND_TREE_NODE_CF.

Each row in the FND_TREE_NODE_CF table represents a path in a tree, and can hold a maximum of 32 nodes. The rows are arranged from leaf to root.

### Figure 20-83  Leaf-to-Root Order

- **Continent level**
  - North America
- **Country level**
  - USA
- **State level**
  - CA
- **City level**
  - LA
  - SF
  - D.C.
The following table shows the results after flattening.

<table>
<thead>
<tr>
<th>Dep0</th>
<th>Dep1</th>
<th>Dep2</th>
<th>Dep3</th>
<th>Dep4</th>
<th>...</th>
<th>Dep31</th>
</tr>
</thead>
<tbody>
<tr>
<td>LA</td>
<td>CA</td>
<td>USA</td>
<td>North America</td>
<td>Null</td>
<td>...</td>
<td>Null</td>
</tr>
<tr>
<td>SF</td>
<td>CA</td>
<td>USA</td>
<td>North America</td>
<td>Null</td>
<td>...</td>
<td>Null</td>
</tr>
<tr>
<td>DC</td>
<td>Null</td>
<td>USA</td>
<td>North America</td>
<td>Null</td>
<td>...</td>
<td>Null</td>
</tr>
</tbody>
</table>

Note:
All fields not containing nodes will be filled with Null.

20.14.5 About Trees Auditing and Flattening of Oracle Enterprise Scheduler Service Job Submissions

For trees with thousands of nodes, the auditing and flattening process for tree versions can take a very long time. To shorten the process, you can submit Oracle Enterprise Scheduler Service jobs using the Oracle Fusion Middleware Extensions for Applications (Applications Core) Setup UI.

From the UI, you can select any tree version on the fly and choose to do the auditing and flattening either "Online" or "Offline". The "Online" option triggers the process in synchronous mode. The "Offline" option allows you to audit or flatten without having to write additional code or create a custom job. It submits a PL/SQL API call that invokes the trees Oracle Enterprise Scheduler Service program for performing the flattening or auditing based on selected options.

Online/Offline Options

Online/offline options are available on the Manage Tree and Tree Versions page. The figures that follow show the Online/Offline options and confirmation dialog in the Applications Core Setup UI.

Figure 20-84  Trees Auditing: Online/Offline Options
20.14.6 About Configuring Oracle Enterprise Scheduler Service for Trees Auditing and Flattening

Configuring Oracle Enterprise Scheduler Service to audit and flatten trees involves the following tasks:

- **Verifying Your Application for Configuration**
- **Implementing the Trees Jobs**
- **Understanding How to Flatten Concurrent Programs**
- **Implementing the Concurrent Programs for the Trees Jobs**
- **Configuring Security for the Jobs**
- **Testing and Verifying**

### 20.14.6.1 Verifying Your Application for Oracle Enterprise Scheduler Configuration

Do the following:

- Ensure that you have a hosting Oracle Enterprise Scheduler Service application for carrying the trees jobs. For example, for Oracle Fusion Human Capital Management, it is the HcmESS application; for Oracle Fusion Financials, it is the FinancialESS application, which is being centrally hosted.

- Ensure that the Setup application consuming the trees taskflow has been configured properly for Oracle Enterprise Scheduler Service so that it can refer the Central Oracle Enterprise Scheduler Service application.

- Prior to uptaking any trees jobs, install Oracle Enterprise Scheduler Service and validate that you are able to deploy the two applications referenced in the first bullet item.

### 20.14.6.2 Implement the Trees Jobs

There are two `.xml` files for trees jobs: one for auditing and one for flattening. 

*Example 20-7* and *Example 20-8* show sample jobs.

Do the following to audit and flatten jobs:
1. Extract the jobs from $BUILD_HOME/jdeveloper/oaext/applcore/trees/Trees-SampleJob.xml to some writeable local file.

2. Open AuditJob.xml, found in essmeta/oracle/apps/ess/fnd/applcore/trees/Jobs and set <parameter name="SYS_effectiveApplication" data-type="string">EarFinLedgerEss</parameter> and <parameter name="SYS_product" data-type="string">EarFinLedgerEss</parameter> to reflect the correct application name for your Central Oracle Enterprise Scheduler Service application. For example, for Oracle Human Capital Management it can be set as something like HCMEss, and Oracle Financials can be set to FinancialEss per their Central Oracle Enterprise Scheduler Service hosting application name.

3. Repeat Steps 1 and 2 for FlattenJob.xml, which also can be found in essmeta/oracle/apps/ess/fnd/applcore/trees/Jobs.

4. Copy both jobs from essmeta/oracle/apps/ess/fnd/applcore/trees/Jobs to your hosting Oracle Enterprise Scheduler Service application.

5. Ensure that your hosting Oracle Enterprise Scheduler Service project compiles without errors, and that both your existing MAR profile and generated MAR picks up the jobs.

Example 20-7 Sample Audit Job

```xml
<?xml version = '1.0'?>
<job-definition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="AuditJob" job-type="/oracle/as/ess/ext/JobType/JavaJobType.xml" xmlns="http://xmlns.oracle.com/scheduler">
  <display-name>AuditJob</display-name>
  <description>Audit for Trees and Tree Versions</description>
  <parameter-list>
    <parameter name="SYS_product" data-type="string">EarFinLedgerEss</parameter>
    <parameter name="SYS_className" data-type="string">oracle.apps.fnd.applcore.trees.ess.program.AuditCP</parameter>
    <parameter name="SYS_effectiveApplication" data-type="string">EarFinLedgerEss</parameter>
  </parameter-list></job-definition>
```

Example 20-8 Sample Flatten Job

```xml
<?xml version = '1.0'?>
<job-definition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="FlattenJob" job-type="/oracle/as/ess/ext/JobType/JavaJobType.xml" xmlns="http://xmlns.oracle.com/scheduler">
  <display-name>FlattenJob</display-name>
  <description>Flattening for Trees and Tree Versions</description>
  <parameter-list>
    <parameter name="SYS_product" data-type="string">EarFinLedgerEss</parameter>
    <parameter name="SYS_className" data-type="string">oracle.apps.fnd.applcore.trees.ess.program.Flattening</parameter>
  </parameter-list></job-definition>
```
20.14.6.3 Understand How to Flatten Concurrent Programs

`FlattenCP.java` is the concurrent program provided for flattening Oracle Enterprise Scheduler Service jobs. To support various flattening requirements, the program accepts the parameters in the following table as part of Oracle Enterprise Scheduler Service request.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>TreeStructureCode</td>
<td>Mandatory parameter to pass the tree structure code value of the hierarchy for which flattening needs to be scheduled</td>
</tr>
<tr>
<td>TreeCode</td>
<td>Mandatory parameter to pass the tree code value of the hierarchy for which flattening needs to be scheduled</td>
</tr>
<tr>
<td>TreeVersionId</td>
<td>Mandatory parameter to pass the tree version ID value of the hierarchy for which flattening needs to be scheduled</td>
</tr>
<tr>
<td>FlatteningMode</td>
<td>Mandatory parameter to pass the type of flattening requested. Accepted values are the following:</td>
</tr>
<tr>
<td></td>
<td>• ROW: To request row flattening only</td>
</tr>
<tr>
<td></td>
<td>• COLUMN: To request column flattening only</td>
</tr>
<tr>
<td></td>
<td>• BOTH: To request both row and column flattening simultaneously</td>
</tr>
<tr>
<td>Force</td>
<td>Mandatory parameter to indicate if force flattening is requested. (Values: Y or N)</td>
</tr>
</tbody>
</table>

20.14.6.4 Implement the Concurrent Programs for the Trees Jobs

For your Central hosting Oracle Enterprise Scheduler Service application, open `weblogic-application.xml` and ensure that you have referenced the Oracle Fusion Middleware Extensions for Applications / Oracle Enterprise Scheduler Service library.

```xml
<library-ref>
  <library-name>oracle.applcore.ess</library-name>
</library-ref>
```

**Note:**

It is assumed that all trees being audited through the Central Oracle Enterprise Scheduler Service hosting application have access to the `.jar` files that contain the related view objects for the data source. Otherwise, you will encounter "class not found" exceptions during audit or flattening.

20.14.6.5 Configure Security for the Jobs

It is necessary to add the metadata security permission for the two jobs you packaged in your hosted Oracle Enterprise Scheduler Service application.

Add `jazn-data.xml` entries, like the ones shown below, for these trees jobs.

```xml
<permission>
  <class>oracle.as.scheduler.security.MetadataPermission</class>
  <name>oracle.apps.ess.fnd.applcore.trees.JobDefinition.FlattenJob</name>
</permission>
```
20.14.6.6 Test and Verifying

You now should be able to launch the Trees Summary page and perform the following operations:

- Schedule an audit for a tree or tree version and view the results of the audit.
- Activate a tree version by setting its status to “Active.” (If an audit is needed, the Audit page will open; otherwise, the tree version will be activated.)
- Schedule flattening for rows and columns.

You can schedule flattening for rows and columns in both incremental and force modes, as shown in the following figure.

![Figure 20-87 Schedule Force Flattening Option](image)

20.14.7 About Using Trees Business Events

A business event typically is a one-way, fire-and-forget, asynchronous way to send a notification of a business occurrence. You can raise business events when a situation of interest occurs. The Tree Management infrastructure provides create, update, and delete business events on tree structures, trees and tree versions. The event definitions are available in `$MW_HOME/jdeveloper/jdev/oaext/events/Trees-Model-Events.jar`.

The following table includes details of the create, update, and delete events.

<table>
<thead>
<tr>
<th>Entity</th>
<th>Event Name</th>
<th>Condition</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree Structure</td>
<td>TreeStructureCreateEvent</td>
<td>Create</td>
<td>TreeStructureCode</td>
</tr>
</tbody>
</table>
### Table 20-29  (Cont.) Trees Business Events

<table>
<thead>
<tr>
<th>Entity</th>
<th>Event Name</th>
<th>Condition</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree Structure</td>
<td>TreeStructureUpdateEvent</td>
<td>Update</td>
<td>TreeStructureCode</td>
</tr>
<tr>
<td>Tree Structure</td>
<td>TreeStructureDeleteEvent</td>
<td>Delete</td>
<td>TreeStructureCode</td>
</tr>
<tr>
<td>Tree</td>
<td>TreeCreateEvent</td>
<td>Create</td>
<td>TreeStructureCode, TreeCode</td>
</tr>
<tr>
<td>Tree</td>
<td>TreeUpdateEvent</td>
<td>Update</td>
<td>TreeStructureCode, TreeCode</td>
</tr>
<tr>
<td>Tree</td>
<td>TreeDeleteEvent</td>
<td>Delete</td>
<td>TreeStructureCode, TreeCode</td>
</tr>
<tr>
<td>Tree Version</td>
<td>TreeVersionCreateEvent</td>
<td>Create</td>
<td>TreeStructureCode, TreeCode, TreeVersionId</td>
</tr>
<tr>
<td>Tree Version</td>
<td>TreeVersionUpdateEvent</td>
<td>Update</td>
<td>TreeStructureCode, TreeCode, TreeVersionId</td>
</tr>
<tr>
<td>Tree Version</td>
<td>TreeVersionDeleteEvent</td>
<td>Delete</td>
<td>TreeStructureCode, TreeCode, TreeVersionId</td>
</tr>
<tr>
<td>Tree Node</td>
<td>TreeNode Created</td>
<td>New node added to tree (includes value, range and referenced tree nodes)</td>
<td>TreeStructureCode, TreeCode, TreeVersionId, TreeNodeId</td>
</tr>
<tr>
<td>Tree Node</td>
<td>TreeNode Deleted</td>
<td>Node removed from tree (includes value, range and referenced tree nodes)</td>
<td>TreeStructureCode, TreeCode, TreeVersionId, TreeNodeId</td>
</tr>
<tr>
<td>Tree Node</td>
<td>TreeNode Updated</td>
<td>Node updated in tree (includes value, range and referenced tree nodes)</td>
<td>TreeStructureCode, TreeCode, TreeVersionId, TreeNodeId</td>
</tr>
<tr>
<td>Tree Node</td>
<td>TreeNode Moved</td>
<td>Node moved within tree (includes value, range and referenced tree nodes)</td>
<td>TreeStructureCode, TreeCode, TreeVersionId, TreeNodeId</td>
</tr>
</tbody>
</table>

### Related Links

The following document provides more information about the topic discussed in this section:

- For more information, see *Using Business Events and the Event Delivery Network* in *Developing SOA Applications with Oracle SOA Suite*.

### 20.14.8 About Using WLST Commands for Flattening

WebLogic Scripting Tool (WLST) is a command line scripting interface that you can use to create, manage, and monitor WebLogic Server domains.
Tree management provides the data flattening feature that improves query performance against the hierarchical data. It supports two types of flattening: column flattening and row flattening. Column flattening optimizes the parent-child information for run-time performance by storing additional column in a table for all parents of a child. Row flattening optimizes parent-child information for run-time performance by storing additional rows in a table (as compared to just normalized parent-child rows) to instantly find all descendants to a parent value, without initiating a CONNECT BY SQL statement.

WLST commands are to be executed from the `wlst` prompt. Execute the shell script located at `$JDEV_HOME/oracle_common/common/bin` to navigate to the `wlst` prompt.

```
cd $JDEV_HOME/oracle_common/common/bin/
sh wlst.sh
```

The `treeManagement.py` Python Script defines the WLST commands that can be executed at the WLST prompt. Each command corresponds to a method in the python script. The python script is available at `$ADE_VIEW_ROOT/atgpf/applcore/etc/wlst/`.

A package `oracle.apps.fnd.applcore.trees.mbean` is added to the Trees ViewController Layer. This package includes the following files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TreeFlatteningMBeanLifeCycleCallBack.java</td>
<td>Manages registration and de-registration of Trees MBean</td>
</tr>
<tr>
<td>TreeMBeanUtil.java</td>
<td></td>
</tr>
<tr>
<td>TreeFlatteningMBean.java</td>
<td>Defines the signature and implements the java based code logic for all WLST commands</td>
</tr>
<tr>
<td>TreeFlatteningMBeanImpl.java</td>
<td></td>
</tr>
<tr>
<td>FlatteningBean.java</td>
<td>Bean Data Structure for the object that stores flattening results for all individual Tree Versions</td>
</tr>
<tr>
<td>TreeFlatteningXMLReport.java</td>
<td>Outlines the design and generates the output XML Report for the WLST commands</td>
</tr>
<tr>
<td>TreeMBeanUtil.java</td>
<td>Serves as the supporting utility and resource files</td>
</tr>
<tr>
<td>UIModelConstants.java</td>
<td></td>
</tr>
<tr>
<td>UIModelMsgBundle.java</td>
<td></td>
</tr>
</tbody>
</table>

The defined contextual parameter in `TreeDeployMBeanLifeCycleCallBack.java` and the class need to be registered as a listener in the `web.xml` file of the Fnd Setup application.

**Contextual Parameter:**

```
<context-param>
  <param-name>oracle.apps.fnd.internal.trees.mbean.ENABLED</param-name>
  <param-value>true</param-value>
</context-param>
```

**Listener Class:**

```
<listener>
  <listener-class>
```
Ensure the following before you execute the WLST commands for flattening:

- Integrated WebLogic Server is up and running.
- Connection is established to the WebLogic Server.
- Connection is established to the Trees MBean package.
- Each WLST command is defined.

The MBean package exposes the following set of APIs:

- flattenAll
- flattenTreeStructure
- flattenTree
- flattenTreeVersion
- flattenTreeVersionAsync
- flattenTreeVersionId
- forceFlattenTreeVersion
- auditTreeVersion
- auditTreeVersionAsync

### 20.14.8.1 How to Invoke Flattening APIs

The flattening APIs exposed by the MBean package can be invoked in two ways:

1. by passing the required parameters.

   **Note:**
   You can pass the parameters in any order.

2. by passing the parameter name and the corresponding value.

   **Note:**
   The order in which you pass the parameter and its value must match with the signature of the API.

### 20.14.8.2 About Using the `flattenAll` API

This API performs incremental flattening of all active Tree Versions under all the available Tree Structures in the application.
Note:
Only active Tree Versions are flattened.

The following table describes the parameters of this API:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>String</td>
<td>Type of flattening to be performed. It can be COLUMN or ROW flattening. By default, this API triggers both types of flattening</td>
<td>Optional</td>
</tr>
</tbody>
</table>

The following is an example of this API:

**Example 20-9 flattenAll API Example**

```python
flattenAll(type='COLUMN')
```

20.14.8.3 About Using the flattenTreeStructure API

This API performs incremental flattening for all active hierarchies for a given Tree Structure.

Note:
Only active Tree Structures are flattened.

The following table describes the parameters of this API:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>treeStructureCode</td>
<td>String</td>
<td>Code of the tree structure on which flattening needs to be performed</td>
<td>Mandatory</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
<td>Type of flattening to be performed. It can be COLUMN or ROW flattening. By default, this API triggers both types of flattening</td>
<td>Optional</td>
</tr>
</tbody>
</table>

The following is an example of this API:

**Example 20-10 flattenTreeStructure API Example**

```python
flattenTreeStructure(treeStructureCode='FND_DEMO_EMP_TS')
```
20.14.8.4 About Using the flattenTree API

This API performs incremental flattening for all active Tree Versions for the specified Tree Code.

Note:
Only active Tree Versions are flattened.

The following table describes the parameters of this API:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>treeStructureCode</td>
<td>String</td>
<td>Code of the tree structure on which flattening needs to be performed</td>
<td>Mandatory</td>
</tr>
<tr>
<td>treeCode</td>
<td>String</td>
<td>Code of the tree on which flattening needs to be performed</td>
<td>Mandatory</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
<td>Type of flattening to be performed. It can be COLUMN or ROW flattening. By default, this API triggers both types of flattening</td>
<td>Optional</td>
</tr>
</tbody>
</table>

The following is an example of this API:

Example 20-11 flattenTree API Example

flattenTree(treeCode='FND_DEMO_EMP_T',treeStructureCode='FND_DEMO_EMP_TS')

20.14.8.5 About Using the flattenTreeVersion API

This API performs incremental flattening for the specified tree version synchronously.

Note:
Only the tree structures that are in active or draft state are flattened.

The following table describes the parameters of this API:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>treeStructureCode</td>
<td>String</td>
<td>Code of the tree structure on which flattening needs to be performed</td>
<td>Mandatory</td>
</tr>
<tr>
<td>treeCode</td>
<td>String</td>
<td>Code of the tree on which flattening needs to be performed</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>
### flattenTreeVersion API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>treeVersionName</td>
<td>String</td>
<td>Name of the tree version that needs to be flattened</td>
<td>Mandatory</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
<td>Type of flattening to be performed. It can be COLUMN or ROW flattening. By default, this API triggers both types of flattening</td>
<td>Optional</td>
</tr>
</tbody>
</table>

The following is an example of this API:

**Example 20-12 flattenTreeVersion API Example**

```java
flattenTreeVersion(treeStructureCode='FND_DEMO_EMP_TS',treeCode='FND_DEMO_EMP_T',treeVersionName='FND Demo Employee Tree Version',type='ROW')
```

### flattenTreeVersionAsync API

This API performs incremental flattening for the specified tree version asynchronously.

**Note:**

Only the tree structures that are in active or draft state are flattened.

The following table describes the parameters of this API:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>treeStructureCode</td>
<td>String</td>
<td>Code of the tree structure on which flattening needs to be performed</td>
<td>Mandatory</td>
</tr>
<tr>
<td>treeCode</td>
<td>String</td>
<td>Code of the tree on which flattening needs to be performed</td>
<td>Mandatory</td>
</tr>
<tr>
<td>treeVersionName</td>
<td>String</td>
<td>Name of the tree version that needs to be flattened</td>
<td>Mandatory</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
<td>Type of flattening to be performed. It can be COLUMN or ROW flattening. By default, this API triggers both types of flattening</td>
<td>Optional</td>
</tr>
</tbody>
</table>

The following is an example of this API:
Example 20-13 flattenTreeVersionAsync API Example
flattenTreeVersion(treeStructureCode='FND_DEMO_EMP_TS',treeCode='FND_DEMO_EMP_T',treeVersionName='FND Demo Employee Tree Version',type='ROW')

20.14.8.7 About Using the flattenTreeVersionId API
This API performs incremental flattening for the specified tree version with the given ID.

The following table describes the parameters of this API:

Table 20-36 flattenTreeVersionId API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>treeStructureCode</td>
<td>String</td>
<td>Code of the tree structure on which flattening needs to be performed</td>
<td>Mandatory</td>
</tr>
<tr>
<td>treeCode</td>
<td>String</td>
<td>Code of the tree on which flattening needs to be performed</td>
<td>Mandatory</td>
</tr>
<tr>
<td>treeVersionId</td>
<td>String</td>
<td>Id of the tree version that needs to be flattened</td>
<td>Mandatory</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
<td>Type of flattening to be performed. It can be COLUMN or ROW flattening. By default, this API triggers both types of flattening</td>
<td>Optional</td>
</tr>
</tbody>
</table>

The following is an example of this API:

Example 20-14 flattenTreeVersionId API Example
flattenTreeVersion(treeStructureCode='FND_DEMO_EMP_TS',treeCode='FND_DEMO_EMP_T',treeVersionId='FNDDemoEmployeeTreeVersion',type='ROW')

20.14.8.8 About Using the forceFlattenTreeVersion API
This API forces flattening on the specified tree version.

Note:
Only the tree versions that are in active or draft state are flattened.

The following table describes the parameters of this API:

Table 20-37 forceFlattenTreeVersion API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>treeStructureCode</td>
<td>String</td>
<td>Code of the tree structure on which flattening needs to be performed</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>
Table 20-37  (Cont.) forceFlattenTreeVersion API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>treeCode</td>
<td>String</td>
<td>Code of the tree on which flattening needs to be performed</td>
<td>Mandatory</td>
</tr>
<tr>
<td>treeVersionName</td>
<td>String</td>
<td>Name of the tree version that needs to be flattened</td>
<td>Mandatory</td>
</tr>
</tbody>
</table>

The following is an example of this API:

**Example 20-15  forceFlattenTreeVersion API Example**

forceFlattenTreeVersion(treeStructureCode='FND_DEMO_EMP_TS',treeCode='FND_DEMO_EMP_T',treeVersionName='FND Demo Employee Tree Version')

20.14.8.9 About Using the **auditTreeVersion** API

This API performs an audit of the specified tree version synchronously.

The following table describes the parameters of this API:

Table 20-38  auditTreeVersion API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>treeStructureCode</td>
<td>String</td>
<td>Code of the tree structure on which auditing needs to be performed</td>
<td>Mandatory</td>
</tr>
<tr>
<td>treeCode</td>
<td>String</td>
<td>Code of the tree on which auditing needs to be performed</td>
<td>Mandatory</td>
</tr>
<tr>
<td>treeVersionName</td>
<td>String</td>
<td>Name of the tree version that needs to be audited</td>
<td>Mandatory</td>
</tr>
<tr>
<td>setActive</td>
<td>Boolean</td>
<td>Flag to indicate that the tree version is in active status after auditing</td>
<td>Optional</td>
</tr>
</tbody>
</table>

The following is an example of this API:

**Example 20-16  auditTreeVersion API Example**

auditTreeVersion(treeStructureCode='FND_DEMO_EMP_TS',treeCode='FND_DEMO_EMP_T',treeVersionName='FND Demo Employee Tree Version',setActive=true)

20.14.8.10 About Using the **auditTreeVersionAsync** API

This API performs an audit of the specified tree version asynchronously.

The following table describes the parameters of this API:
Table 20-39  auditTreeAsyncVersion API Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Datatype</th>
<th>Description</th>
<th>Mandatory/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>treeStructureCode</td>
<td>String</td>
<td>Code of the tree structure on which auditing needs to be performed</td>
<td>Mandatory</td>
</tr>
<tr>
<td>treeCode</td>
<td>String</td>
<td>Code of the tree on which auditing needs to be performed</td>
<td>Mandatory</td>
</tr>
<tr>
<td>treeVersionName</td>
<td>String</td>
<td>Name of the tree version that needs to be audited</td>
<td>Mandatory</td>
</tr>
<tr>
<td>setActive</td>
<td>Boolean</td>
<td>Flag to indicate that the tree version is in active status after auditing</td>
<td>Optional</td>
</tr>
</tbody>
</table>

The following is an example of this API:

Example 20-17  auditTreeVersionAsync API Example

auditTreeVersionAsync(treeStructureCode='FND_DEMO_EMP_TS', treeCode='FND_DEMO_EMP_T', treeVersionName='FND Demo Employee Tree Version', setActive=true)

20.14.9 Understanding XML Report Formats for WLST Commands

A WLST Command Report is generated and printed on the console when a WLST command is executed. The following details are displayed on the console:

- WLST command that is executed
- registration of the MBean
- flattening call invoked
- XML report

Note:
You can redirect the details printed on the console to a file, if required.

An XML Report is generated every time you execute a WLST command. The XML report contains all the results of the flattening APIs invoked. The following examples illustrate the XML report generated in various scenarios.

Example 20-18  WLST Command Report

wls:/DefaultDomain/serverConfig>
flattenTree(treeStructureCode='TEST_VS_BICVO_TS',treeCode='TEST_VSBICVO_TREE')
<FLATTENING_INFO>Trees Flattening Mbean name:oracle.apps.fnd.applcore.trees.mbean:name=TreeFlatten,*<SUCCESS> Connected to Weblogic Server Location changed to domainRuntime tree. This is a read-only tree with DomainMBean as the root.
For more help, use help(domainRuntime)
<FLATTENING_INFO> Type of Flattening : BOTH
<SUCCESS> TreeFlatten MBean found:
oracle.apps.fnd.appcore.trees.mbean:Location=DefaultServer, name=TreeFlatten, type=App
sRuntimeMBean, Application=FndSetup, ApplicationVersion=V2.0
<FLATTENING_INFO> Invoking Mbean : for operation flattenTree
<FLATTENING_INFO> parseXMLForError : returning (""
XML report Start:
</xml version = '1.0' encoding = 'UTF-8'>
<TreesFlatteningReport status="SUCCESS">
<TreeStructure treeStructureCode="TEST_VS_BICVO_TS" treeCodeCount="1" success="1" error="0">
</TreeCode>
</TreeStructure>
</TreesFlatteningReport>
XML Report Ends
<FLATTENING_INFO>Completed flattenTree.

Example 20-19  Success Report

</xml version = '1.0' encoding = 'UTF-8'>
<TreesFlatteningReport status="SUCCESS">
<TreeStructure treeStructureCode="TEST_PLSQL_EXTN_TS" treeCodeCount="2" success="2" error="0">
</TreeCode>
</TreeStructure>
</TreesFlatteningReport>

Example 20-20  Error Report with Flattening Errors but no Exception

<xml version = '1.0' encoding = 'UTF-8'>
<TreesFlatteningReport status="ERROR">
    <TreeStructure treeStructureCode="TEST_PLSQL_EXTN_TS" treeCodeCount="1" success="0" error="1">
        <TreeCode treeCode="TEST_PLSQL_EXTN_TR1" treeVersionCount="3" success="2" error="1">
            <TreeVersion treeVersionName="TEST_PLSQL_EXTN_TR1_V1" treeVersionId="B7598796828D8027E040449895F14B45" flatteningType="ROW" time="4.0secs" status="SUCCESS"/>
            <TreeVersion treeVersionName="TEST_PLSQL_EXTN_TR1_V1" treeVersionId="B7598796828D8027E040449895F14B45" flatteningType="COLUMN" time="0.0secs" status="ERROR">
                ERROR:JBO-FTM-S-0045: Flattening cannot proceed due to the failure of auditing tree version TEST_PLSQL_EXTN_TS, TEST_PLSQL_EXTN_TR1, B7598796828D8027E040449895F14B45.
            </TreeVersion>
            <TreeVersion treeVersionName="TEST_PLSQL_EXTN_TR1_V2" treeVersionId="B7598796829A8027E040449895F14B45" flatteningType="ROW" time="0.0secs" status="SUCCESS"/>
            <TreeVersion treeVersionName="TEST_PLSQL_EXTN_TR1_V2" treeVersionId="B7598796829A8027E040449895F14B45" flatteningType="COLUMN" time="1.0secs" status="SUCCESS"/>
            <TreeVersion treeVersionName="TEST_PLSQL_EXTN_TR1_V3" treeVersionId="B7598796829F8027E040449895F14B45" flatteningType="ROW" time="0.0secs" status="SUCCESS"/>
            <TreeVersion treeVersionName="TEST_PLSQL_EXTN_TR1_V3" treeVersionId="B7598796829F8027E040449895F14B45" flatteningType="COLUMN" time="1.0secs" status="SUCCESS"/>
        </TreeCode>
    </TreeStructure>
</TreesFlatteningReport>

Example 20-21  Error Report with Exception but no Flattening Errors

<xml version = '1.0' encoding = 'UTF-8'>
<TreesFlatteningReport status="ERROR">
    <Exception>null releaseTreeService Exception: Cannot release TreeService Handle</Exception>
    <TreeStructure treeStructureCode="FND_DEMO_EMP_TS" treeCodeCount="1" success="1" error="0">
        <TreeCode treeCode="FND_DEMO_EMP_T" treeVersionCount="1" success="1" error="0">
            <TreeVersion treeVersionName="FND Demo Employee Tree Version" treeVersionId="A43EE0B37975BF52E040449833F06958" flatteningType="ROW" time="0.0secs" status="SUCCESS"/>
            <TreeVersion treeVersionName="FND Demo Employee Tree Version" treeVersionId="A43EE0B37975BF52E040449833F06958" flatteningType="COLUMN" time="0.0secs" status="SUCCESS"/>
        </TreeCode>
    </TreeStructure>
</TreesFlatteningReport>
20.14.10 About Purging Unused Tree Logs

Node operations performed for a tree hierarchy also generate logs in trees tables that identify the changes made during the tree-flattening process. Once the flattening is processed for a hierarchy version, the logs that were generated prior to the last flattening run become obsolete. Over time, these obsolete records can accumulate in the log table, slowing the querying process that occurs during tree duplication or flattening.

You can purge all of these unused or archived logs for all underlying tree and tree versions associated with a selected tree structure by going to **Tree Structure Search UI > Actions > Purge Logs** in the Manage Tree Structures task flow.

**Note:**
You can only purge logs that were generated at least one day prior to the last flattening run for any tree version.
This chapter describes the Oracle Fusion Applications standards and guidelines for working with localization formatting.

- Introduction to Localization Formatting
- Formatting Currency
- Formatting Numbers
- Formatting Date and Timestamp Values
- Formatting Time Zones
- Formatting Numbers, Currency and Dates Using Localization Expression Language Functions
- Implementing Bi-directional Support
- Supporting Mnemonic Keys
- Implementing Localization Formatting in ADF Desktop Integration
- Implementing Localization Formatting in Reports
- Implementing Localization Formatting in ADF Data Visualization Components
- Configuring National Language Support Attributes
- Standards and Guidelines for Localization Formatting

### 21.1 Introduction to Localization Formatting

When developing applications for international users, it is often necessary to format the display of certain location-dependent data. In the context of Oracle JDeveloper and Oracle Application Development Framework, localization requires implementing formatting patterns so as to properly display the data according to local standards.

Currency, dates, numbers and time zones are localized using Oracle ADF Faces components that bind to the attribute to be localized. In some cases, Groovy or Expression Language methods are used to localize a particular UI component.

### 21.2 Formatting Currency

Currency values may be formatted differently, depending on the currency code associated with the value. Each currency code is associated with formatting such as precision and currency symbols.
You can format currency using the default formatting behavior, or by overriding the default formatting. Alternatively, you can format currency on the fly using partial page rendering.

Currency fields are represented by `java.lang.BigDecimal` in entity objects and view objects.

A currency field should always be formatted according to the currency code chosen in the context UI of the transaction.

For example, if the user selects JPY as the currency code from the context UI, then the currency value should be formatted according to the Japanese Yen standard.

There are two implementations to format numerical values according to the corresponding currency code.

- One is the Expression Language function `fnd:currencyPattern(currencyCode)` that takes the currency code input in the context user interface as a parameter. If JPY is input to this Expression Language function, then the number would be formatted according to the Japanese Yen pattern: #,##0;-#,##0 without showing any decimal digits, whereas if the same Expression Language function is used for the US dollar input parameter, the number would be formatted according to the USD pattern: #,##0.00;-#,##0.00. For example, the numerical value 135.6789 that is fetched from the database must be shown as 135.68 USD, whereas the same number in JPY would be shown as 136 JPY.

  The syntax is:
  
  ```
  fnd:currencyPattern(bindingToAmountCurrencyCode)
  ```

- The other implementation to format currency values is the `fnd:currencyPatternWithPrecisionAndSymbol` Expression Language function, which is a more flexible function than `fnd:currencyPattern`. The `fnd:currencyPatternWithPrecisionAndSymbol` function takes these input parameters:
  
  - currency code to be used for formatting
  - number of precision digits required in the formatted number
  - whether the currency code or symbol is to be shown in the formatted number

  The syntax is shown in Example 21-1.

  For example, if JPY is entered as the formatting currency code, and `symbol` is used as the currency code/symbol parameter, and the number of precision digits is set to 1, then the number would be formatted as #,##0 @ - #,##0 @. The numerical value 135.6789 retrieved from the database must be shown as 135.68$ in USD, whereas the same number will need to be shown as 136@ in JPY. (This example considers that the `currencySymbol` parameter has been set to `symbol`). The `currencySymbol` parameter can only take the values `symbol`, `code`, or `none`.

Currency codes are stored in the `FND_Currencies` table located in the Oracle Fusion Middleware Extensions for Applications schema. The currency code determines the format mask for the currency field, including:

**Precision**: Determines the use and number of decimal digits, comma placement, and so on.

**Currency symbol**: Displays the standard symbol for the currency.
The Expression Language function `fnd:currencyPattern()` determines formatting for the currency field using the default precision associated with the currency code. No currency symbol or code is displayed in the user interface. The Expression Language function `fnd:currencyPatternWithPrecisionAndSymbol()` determines formatting for the currency field using an extra parameter for precision. It also uses the currency code to show a currency symbol/code in the user interface. User preferences for grouping and decimal separators are used to format the value in both these functions.

**Example 21-1  Syntax of fnd:currencyPatternWithPrecisionAndSymbol**

```java
fnd:currencyPatternWithPrecisionAndSymbol(
    bindingToAmountCurrencyCode, bindingToAttrNamePrecision,
    bindingToAttrNameCurrencySymbol)
```

21.2.1 How to Format Currency

To format currency, you must first set the view object custom attribute to type `currency` and then select the currency code to be used in formatting the currency field. You can specify the currency code by entering its value explicitly, or by writing a Groovy expression in the transient attribute.

**Before you begin:**

Create an entity object and a view object.

**Example 21-2  Expression that Identifies the Currency Code**

```java
object.currencyCode
```

21.2.1.1 Formatting Currency Values

To format currency values, use the `fnd:currencyPattern()` function to obtain the format pattern of different currency codes. Therefore, a currency field, whether it is used in `inputText` or `outputText`, should be formatted with code similar to that shown here.

```xml
<af:convertNumber type="currency" currencyCode="#{bindingToAmountCurrencyCode}" pattern="#{fnd:currencyPattern(bindingToAmountCurrencyCode)}" />
```

Here, `bindingToAmountCurrencyCode` should be replaced with the actual name of the field that allows users to select a currency code.

If `<af:convertNumber type="currency" pattern="#{applCorePrefs.numberFormatPattern}"/>` is found, then the currency field will be formatted as a number. *This is incorrect.*

If the field requires displaying currency numbers combined with a currency code or currency symbol, the Expression Language function `fnd:currencyPatternWithPrecisionAndSymbol()` is necessary to get the format pattern.

In Example 21-3, the Amount field is formatted according to the currency code set in the `af:convertNumber` tag.

In Example 21-4, the grouping separator and decimal separator are taken from the user's number preferences, and the number of precision digits is set to 2. Also, the symbol shown corresponds to the currency code set in the `af:convertNumber` tag.

The fields Ordered, Total Tax, and Total in Figure 21-1 show how currency values are formatted in Oracle Fusion Applications.
21.2.1.2 What Happens When You Format Currency

When dragging and dropping the view object onto a JSF page, Applications Core generates relevant bindings to the Oracle ADF Faces Number Converter attached to the user interface component.

Example 21-5 shows bindings to the Number Converter.

As the developer, you should change the generated code according to either the fnd:currencyPattern() Expression Language function or the fnd:currencyPatternWithPrecisionAndSymbol() Expression Language function as shown in Example 21-6.

The bindings indicate that the Expression Language method fnd:currencyPattern() is to be used to return the currency code for the value of the attribute.

Example 21-5   Bindings to the Number Converter

Example 21-6   Changing the Generated Code
21.2.1.3 What Happens at Runtime: How Currency Is Formatted

At runtime, Applications Core Technology evaluates the bindings generated during design time to generate the correct currency format mask for the value. The `fnd:currencyPattern()` and `fnd:currencyPatternWithPrecisionAndSymbol()` Expression Language methods return the format mask for a given currency code, accounting for currency precision, currency symbol, and so on.

21.2.2 How to Format Currency and Override the Default Formatting Behavior

You can change the default currency formatting behavior by editing the bindings and implementing a different Expression Language method from the one used in How to Format Currency.

To format currency, overriding the default formatting behavior:

1. Follow the procedure described in How to Format Currency up to step 5.
2. To display currency symbols and override the default currency code precision, use the `fnd:currencyPatternWithPrecisionAndSymbol()` Expression Language method as shown in Example 21-7.

   The `currencySymbol` parameter can take the values `symbol`, `code`, or `none`.

   **Note:**
   The type and currency code are not specified as the formatting pattern is generated by the Oracle FND function, and not by Oracle ADF Faces.

   The Expression Language method `fnd:currencyPatternWithPrecisionAndSymbol()` is to be used to return the currency code, currency symbol, and precision values for the value of the attribute.

   For more information on the `currencyPatternWithPrecisionAndSymbol` method, see the Javadoc for the `oracle.apps.fnd.applcore.common.Preferences` class.

3. To display currency values in a user interface, drag and drop the view object including the currency attribute onto the JSF page.

   **Example 21-7  Bindings to the Number Converter with Specified Precision and Symbol Display**

   ```xml
   <af:outputText label="Amount" value="#{bindingToOrderTotal}">
   <af:convertNumber pattern="#{fnd:currencyPatternWithPrecisionAndSymbol(
     bindingToAmountCurrencyCode, bindingToAttrNamePrecision,
     bindingToAttrNameCurrencySymbol)}" />
   <af:outputText>
   ```

21.2.3 How to Immediately Format Currency Using Partial Page Rendering

Partial page rendering (PPR) can be used to immediately format or validate a currency-based UI component on the client.
Bug 7699549: The properties `autoSubmit=true` and `partialTriggers=<source attr>` are used not only for currencies but also for ordinary number values. This workaround is required when the pattern property is specified for Trinidad NumberConverter as Trinidad NumberConverter does not support client-side conversion by Javascript when the pattern property is specified. This workaround is only required for Trinidad NumberConverter, and not for DateTimeConverter. This workaround is not required for trinidad DateTimeConverter as client-side Javascript conversion is supported even when specifying a pattern property.

Once this fix has been implemented, the `autoSubmit` and `partialTriggers` workaround will no longer be necessary and should be removed.

To immediately format currency using partial page rendering:

- For the UI component with a currency type value, do the following:
  1. Set the `autoSubmit` attribute to `true`.
  2. Set the `partialTriggers` attribute to the ID of the current UI component, as shown in Example 21-8.

```
Example 21-8 Formatting Currency Using Partial Page Rendering

<af:inputText label="fld1" id="fld1" value="#{simplebean.fld1}"
  autoSubmit="true" partialTriggers="fld1">
  <af:convertNumber pattern="#,#00.00" />
</af:inputText>
```

### 21.2.4 What Happens When You Format Currency

When formatting currency, Applications Core automatically generates a transient view object attribute with the name `attributeNameCurrencyCode`. This transient attribute identifies the currency code to be used when formatting the currency field.

When dragging and dropping the view object onto a JSF page, Applications Core generates relevant bindings to the Oracle ADF Faces Number Converter attached to the user interface component as shown in Example 21-9.

The bindings indicate that the Expression Language method `fnd:currencyPattern()` is to be used to return the currency code for the value of the attribute.

```
Example 21-9 Bindings to the Number Converter

<af:outputText label="Amount" value="#{bindingToOrderTotal}"
  currencyCode="#{bindingToAmountCurrencyCode}"
  pattern="#{fnd:currencyPattern(bindingToAmountCurrencyCode)}" />
<af:outputText>
```

### 21.2.5 What Happens at Runtime: How Currency Is Formatted

At runtime, Applications Core evaluates the bindings generated during design time to generate the correct currency format mask for the value. The `fnd:currencyPattern()` and `fnd:currencyPatternWithPrecisionAndSymbol()` Expression Language methods return the format mask for a given currency code, accounting for currency precision, currency symbol, and so on.
21.3 Format Numbers

All numerical values need to be formatted correctly when they are presented to a user in Oracle Fusion Applications. Users expect that they can enter numerical values based on their formatting preferences, which include such things as options to change the grouping and the decimal separators. Number formatting preferences also let users choose the number format mask that will be used to display the number. For instance, a user might choose that the grouping separator has to be shown once every three digits with a maximum of three decimal digits (#,##0.###). An example is a value of 1234.5 in the database that might have to be displayed as 1,234,500 or 1,234.500 or 1.234,500.

21.3.1 How to Format Numbers

Before you begin:

Create an entity object and a view object with number fields, including any of these attribute types:


The number one thousand two hundred and thirty four point five six can display in any of the following formats:

- 1,234.56
- 1’234.56
- 1’234,56
- 1.234,56
- 1234.56 (ISO standard)
- 1234,56
- 1 234.56
- 1 234,56

View objects and entity objects can be formatted so as to display date and number data in accordance with local standards. The ISO standard is typically used when it is not desirable to use local standards.

21.3.1.1 Formatting Decimal Numbers

Decimal output fields should be formatted according to the user’s number formatting preferences. Use the <af:convertNumber pattern...> entry to retrieve the number formatting pattern from the applCorePrefs bean, as shown in Example 21-10.

Input decimal fields should be formatted according to the user’s number preferences. In Example 21-11, use the <af:convertNumber pattern...> entry to retrieve the number formatting pattern from the applCorePrefs bean.

The Threshold Start field in Figure 21-2 is an example of a number field that is formatted according to the user’s number formatting preferences.
When a decimal number is to be shown as a part of a larger string object on the user interface, the fn:formatNumber and fn:formatNumber2 Expression Language functions provided in the applCore library is necessary to format such numbers. When either of these functions is used, the user’s number formatting preferences are implemented according to applCorePrefs.numberFormatPattern.

The syntaxes of the two Expression Language functions are:

- fn:formatNumber(java.lang.Number decimalValueToBeFormatted)
- fn:formatNumber2(java.lang.Number decimalValueToBeFormatted, int maximumNumberOfFractionDigits)

Example 21-12 shows how to use these two functions for decimal number formatting.

As shown in Example 21-12, the variable ActiveResponsesChangeAmt will be formatted according to the user’s number preferences according to applCorePrefs.numberFormatPattern. This formatted number is then concatenated to the externalized string FavorablyIncreasedbyVALUE and displayed to the user.

The fn:formatNumber2 function has the advantage of explicitly specifying the maximum number of fraction digits in the formatted number, as shown in Example 21-13.

Here, the text attribute of the af:showDetailItem tag includes a number that must be formatted before being displayed. Therefore, the fn:formatNumber2 Expression Language function has been used to format the number to be added to the SubmittedValues string. Here, the second parameter has been set to 2 and indicates that a maximum of two fractional digits can be shown in the decimal number. So, for example, after formatting, the number 1234.56789 that is retrieved from the database will be displayed as SubmittedValues: 1,234.57.

Example 21-10  Formatting Decimal Output Fields

```html
<af:outputText value="${bindings.FromValue.inputValue}" id="ot19">
<af:convertNumber pattern="${applCorePrefs.numberFormatPattern}"/>
</af:outputText>
```

Example 21-11  Formatting Input Decimal Fields

```html
<af:inputText value="${row.bindings.FromValue.inputValue}"
label="${bindings.PerformanceThreshold21hints.FromValue.label}"
required="${bindings.PerformanceThreshold21hints.FromValue.mandatory}"
columns="${bindings.PerformanceThreshold21hints.FromValue.displayWidth}"
maximumLength="${bindings.PerformanceThreshold21hints.FromValue.precision}"
shortDesc="${bindings.PerformanceThreshold21hints.FromValue.tooltip}"
id="inputText9"
autoSubmit="true">
```
Example 21-12  Decimal Number Formatting

Example 21-13  Using the fnd:formatNumber2 Function

21.3.1.2 Formatting Integer Numbers

In Figure 21-3, Point 1 is an example of an input integer field that is formatted according to the user's number preferences without showing the decimal separator or any precision digits.
When an integer number is to be shown as a part of a larger string object on the user interface, the \texttt{fnd:formatNumber2} function provided in the applCore library is necessary to format such numbers. When this function is used with the maximum number of precision digits set to 0, the user's number formatting preferences are used from \texttt{applCorePrefs.numberFormatPattern} with no precision digits.

The syntax of the \texttt{fnd:formatNumber2} function is as follows:

\begin{verbatim}
\texttt{fnd:formatNumber2(}java.lang.Number integerValueToBeFormatted, int maximumNumberOfFractionDigits\texttt{)}
\end{verbatim}

The \texttt{fnd:formatNumber2} function can be used to format integer numbers by explicitly specifying the maximum number of fraction digits as 0 in the formatted number. \textbf{Example 21-16} shows how to use the \texttt{fnd:formatNumber2} function for integer number formatting.

Here, the text attribute of the \texttt{af:showDetailItem} tag includes a number that must be formatted before being displayed. The \texttt{fnd:formatNumber2} function is used to format the integer to be added to the \texttt{SubmittedInvoices} string. The second parameter has been set to 0 and indicates that no fractional digits can be shown in the formatted number. For example, after formatting, the number 1234 that is retrieved from the database will be displayed as \texttt{SubmittedInvoices: 1,234}.

\textbf{Example 21-14} \quad Formatting Integer Numbers

\begin{verbatim}
<af:outputText value="#{bindings.integerQuantity.inputValue}" id="ot19">
<af:convertNumber pattern="#{applCorePrefs.integerFormatPattern}"/>
</af:outputText>
\end{verbatim}

\textbf{Example 21-15} \quad Formatting Input Integer Fields

\begin{verbatim}
<af:inputText value="#{bindings.Point1.inputValue}"
label="#{bindings.Point1.hints.label}"
required="#{bindings.Point1.hints.mandatory}"
columns="#{bindings.Point1.hints.displayWidth}"
maxLength="#{bindings.Point1.hints.precision}"
shortDesc="#{bindings.Point1.hints.tooltip}"
inlineStyle="width:100%; border-color:InactiveBorder; border-width:thin;"
\end{verbatim}
21.3.1.3 Formatting ID Numbers

An ID field is a number field that uniquely identifies an object. Typical examples of ID fields include TransactionID, Sequence Number, and Item Number. Such ID output fields should be formatted without using the user's number formatting preferences. That is, ID output fields should not display decimal precision digits or grouping separators. For example, a student's admission number should be displayed as 14567 instead of 14,567.00. The `<af:convertNumber pattern...` entry in Example 21-17 retrieves the number formatting pattern from the `applCorePrefs` bean.

Note:

ID fields always should be aligned to the left. As a developer, you must not hardcode the align attribute of the column attribute of an `af:table`, or the `contentStyle` attribute of the `af:outputText` tag displaying an ID field. By default, Oracle ADF supports start alignment that is appropriate for ID fields.

The field Sales Order Line in Figure 21-4 corresponds to the output ID field shown in Example 21-17.

Figure 21-4 Sample Output ID Field

Input integer fields should be formatted according to the user's number preferences without showing any decimal precision digits or grouping separators. The `<af:convertNumber pattern...` entry in Example 21-18 indicates that the number formatting pattern is being retrieved from the `applCorePrefs` bean.
Format Numbers

The Sequence field in Figure 21-5 corresponds to the input ID field shown in
Example 21-6.
Figure 21-5

Example Input ID Field

The fnd:formatNumber() and fnd:formatNumber2() functions must not be used
for ID fields appearing in strings, because they introduce a grouping separator in
number fields, which should not be displayed in ID fields. In such cases, output ID
fields should be treated as normal text fields.
Example 21-17

Formatting ID Numbers

<af:column headerText="#{bindings.Deliveries.hints.SalesOrderLine.label} id="c9" >
<af:outputText value="#{row.bindings.SalesOrderLine.inputValue}"
label="#{bindings.Deliveries.hints.SalesOrderLine.label}" id="outputText3">
<af:convertNumber pattern="#{applCorePrefs.numericCodeFormatPattern}"/>
</af:outputText>
</af:column>

Example 21-18

Formatting Input Integer Fields

<af:inputText value="#{bindings.SequenceNumber.inputValue}"
label="#{bindings.SequenceNumber.hints.label}"
required="#{bindings.SequenceNumber.hints.mandatory}"
columns="#{bindings.SequenceNumber.hints.displayWidth}"
id="inputText1"
partialTriggers="asgStat"
helpTopicId="InvCoreSetup_755E15370133C364E040D30A688161D5V000"
contentStyle="text-align:start"
autoSubmit="true">
<f:validator binding="#{bindings.SequenceNumber.validator}"/>
<af:convertNumber pattern="#{applCorePrefs.numericCodeFormatPattern}"/>
</af:inputText>

21.3.1.4 How to Format Numbers in Hyperlinks
In Oracle Fusion Applications, hyperlinks are represented by af:commandLink and
af:goLink. All numbers found in the labels of hyperlinks in the user interface must
be formatted according to the user's number preferences. The number might either
have to be formatted according to numberFormatPattern if the number value is
decimal in nature, integerFormatPattern if the number value is integer only, and
need not be formatted at all if the value corresponds to an ID value. Code to format
numbers in hyperlinks is shown in Example 21-19.
Because af:convertNumber is not a valid child tag of af:commandLink, the best
way to format a number on a link is by making the text attribute of the
af:commandLink tag empty and then adding an af:outputText tag as a child tag
that displays the formatted numerical value. In this case, the number is an integer
(java.lang.Integer), so the pattern applCorePrefs.integerFormatPattern has
been used. In cases of decimal numbers (java.math.bigDecimal),
applCorePrefs.numberFormatPattern has to be used, while in cases of ID fields
(java.lang.Long), af:convertNumber can be removed and the field can be honored

21-12 Oracle Fusion Applications Developer's Guide


as plain text. The parent hyperlink tag can be either `af:commandLink` or `af:goLink`.

Similarly, numbers that are found on links represented by the `af:goLink` tag in Oracle Application Development Framework (Oracle ADF) should be formatted according to the user's number formatting preferences using the Preferences bean. The procedure to be followed is very similar to that of the `af:commandLink` tag. Sample code for number formatting in `af:goLink` is shown in Example 21-20.

Again, `af:goLink` cannot have an `af:convertNumber` as its child tag, so the text attribute of the `goLink` tag should be made empty and `af:outputText` should be added as a child tag of `af:goLink`. The `af:convertNumber` then can be added as a subtag of the `outputText` tag. The pattern attribute can be changed for this `af:convertNumber` tag, according to whether the number is decimal or an integer. If the number represents an ID field, no number formatting is needed.

Another way of formatting the text in `af:commandLink` and `af:goLink` is by using the Expression Language functions `fnd:formatNumber` and `fnd:formatNumber2` in their text attributes. Sample code to format a number in a `commandLink` using the `fnd:formatNumber2` Expression Language function is shown in Example 21-21.

Here, the second parameter has been set to 0 so that no precision digits are shown. This is the same as using

```
(pattern="#{applCorePrefs.integerFormatPattern}" as used in Example 21-19). The only added feature in this example is that there is no need to add a child `af:outputText` tag for number formatting.
```

Similarly, `af:goLink` can also be formatted using the `fnd:formatNumber` and `fnd:formatNumber2` Expression Language functions. An example of formatting the `af:goLink` shown in Example 21-20 using the `fnd:formatNumber` function is shown in Example 21-22:

```
Here, the `fnd:formatNumber` Expression Language function has been used to format the value in the `af:goLink`. This is the same as using pattern="#{applCorePrefs.numberFormatPattern}" as shown in Example 21-20.
```

Example 21-19  Formatting Numbers in Hyperlinks

```
<af:commandLink id="outputText6" text="" partialSubmit="true" immediate="true" action="#{LocalAreaHandlerBean.showInventory}"
<af:outputText value="#{row.InvPendingCount}" id="ot1">
    <af:convertNumber type="number" pattern="#{applCorePrefs.integerFormatPattern}"/>
</af:outputText>
</af:commandLink>
```

Example 21-20  Formatting a Number in `af:goLink`

```
<af:goLink text="" id="g1l" destination="#{row.children[0].FileUrl}"
<af:outputText id="id1" value="#{bindings.fromValue.inputValue}"
    <af:convertNumber type="number" pattern="#{applCorePrefs.numberFormatPattern}"/>
</af:outputText>
</af:goLink>
```

Example 21-21  Formatting a Number in a `commandLink`

```
<af:commandLink id="outputText6" text="#{fnd:formatNumber2(row.InvPendingCount,0)}"
    partialSubmit="true" immediate="true"
```

Work with Localization Formatting 21-13
Example 21-22  Formatting af:goLink Using fnd:formatNumber

21.3.1.5 How to Format Percentage Values

A percentage value maps to java.math.BigDecimal in entity object and view object files and NUMBER(n, m) (where m! = 0) in underlying database columns.

Such percentage values in output fields must be displayed in a locale-sensitive format. For example, 75.68% in en_US (Language: English Territory: United States) should be displayed as 75,68 % (there is a space between the number and the percentage sign) in the fr_FR (Language: French Territory: France) locale.

The number and percentage sign of a percentage value should always be displayed in a user preferred format in Oracle Fusion Applications. Sample code that can display percentage values is shown in Example 21-23.

Here, maxFractionDigits can be determined by the underlying database column type using this formula:

maxFractionDigits = The scale of database Column Data Type - 2

For example, if the database column data type is NUMBER(8,4), maxFractionDigits can be “2”. If “The scale of database Column Data Type - 2” <= 0, attribute maxFractionDigits should not be added to the convertNumber tag.

This formula arises from the fact that a percentage value like 1% usually is stored in the database column as 0.01. Hence, the maximum number of fraction digits in the percentage value to be displayed on the user interface will always be 2 less than the scale of the database column.

In Oracle Fusion Applications, you must not hardcode the % sign in the value attribute of the outputText tag, because this makes the percentage values locale-insensitive.

The Percent of Project Work Complete field in Figure 21-6 corresponds to the output percentage field shown in Example 21-23.

Figure 21-6    Results of Output Percentage Field

A percentage value in an input field generally is formatted exactly like the output percentage field. Sample code to format input percentage fields is shown in Example 21-24.
Example 21-23  Displaying Percentage Values

```af:outputText value="#{row.PercentComplete != null ? (row.PercentComplete * 0.01): row.PercentComplete}"
   id="outputText5">
   <af:convertNumber minFractionDigits="2" maxFractionDigits="2"
      type="percent"/>
   <af:showPopupBehavior triggerType="click"
      popupId=":::progressTrendPopup"
      alignId="rowHeadCol"/>
</af:outputText>

Example 21-24  Formatting Input Percentage Fields

```af:inputText value="#{bindings.PercentQuantity.inputValue}"
   label="#{bindings.PercentQuantity.hints.label}"
   autoSubmit="true"
   id="inputText10"
   <f:validator binding="#{bindings.PercentQuantity.validator}"/>
   <af:convertNumber type="percent" minFractionDigits="2"
      maxFractionDigits="2"/>
</af:inputText>

21.3.2 How to Format Numeric IDs and Integers

Not all number values should be formatted using the numberFormatPattern property. For example, a numeric identifier should always be a series of numbers with no locale-sensitive punctuation. The applCorePrefs managed bean provides the numericCodeFormatPattern property to format a numeric code in the same way for all users.

Example 21-25 shows the bindings to the property numericCodeFormatPattern.

The integerFormatPattern property enables formatting an integer, as shown in Example 21-26. No fractions are printed.

Example 21-25  Bindings to the numericCodeFormatPattern Property

```af:convertNumber pattern="#{applCorePrefs.numericCodeFormatPattern}"/>

Example 21-26  Bindings to the integerFormatPattern Property

```af:convertNumber pattern="#{applCorePrefs.integerFormatPattern}"/>

21.3.3 What Happens When You Format Numbers

All number formatting patterns default to the formats described in How to Format Numbers.

When dragging and dropping a view object containing an attribute of type java.math.BigDecimal or java.lang.Integer, or java.lang.Long, Applications Core generates code that binds to the numberFormatPattern property in the applCorePrefs managed bean as shown in Example 21-27.

Example 21-27  Bindings to the numberFormatPattern Property

```af:convertNumber pattern="#{applCorePrefs.numberFormatPattern}"/>
21.3.4 What Happens at Runtime: How Numbers Are Formatted

At runtime, the bindings generated at design time are executed. Numbers are displayed according to user preferences for number formatting patterns, such as 1,234.567.

21.3.5 Standards and Guidelines

The following standards and guidelines apply to formatting dates and numbers:

- All date only fields must be represented only by java.sql.Date data types.
- When a value bound to a field is date only, of type java.sql.Date, make sure not to set the time zone to af:convertDateTime.

21.4 Format Date and Timestamp Values

All date values must be formatted correctly when they are presented to a user. Users expect to input values according to their date formatting preferences. For instance, a value of 13 August, 2011 in the database might have to be displayed as 08/13/2011 or 2011.8.13. A similar example of a date-time value is a value of 26 July 2011, 2:00:00 PM might have to be displayed as 14:00 07/26/2011 or 2011.7.26 02:00:00 PM.

21.4.1 How to Format Dates and Timestamp Values

Before you begin:

Create an entity object and a view object with date fields, including either of these attribute types: java.sql.Date or java.sql.Timestamp.

21.4.1.1 Formatting Dates

Dates without time values (called dates) are mapped to java.sql.Date in the entity object/view object layer. These fields should be formatted according to the user's date formatting preferences.

There are two options for formatting dates using the applCorePrefs bean:

- **dateFormatPattern**: This property formats the date according to user preferences, within a two digit range spanning one hundred years (for example, 1950-2049).
- **longDateFormatPattern**: This property formats the date according to user preferences, within a four digit range. For example, the user-specified date format yy/dd/mm is rendered yyyy/dd/mm. This is useful for applications that require a four digit format for rendering years.

Example 21-28 shows the use of the date formatting pattern.

Both the filter for the date and the date field should be formatted according to the user's date formatting preferences.

The values for the Due Date field in Figure 21-7 are examples of how a date value is formatted according to the user's preferences in Oracle Fusion Applications.
You can also use the property `applCorePrefs.longDateFormatPattern` to format a date value. The property `applCorePrefs.longDateFormatPattern` formats a date value using nearly the same pattern as `applCorePrefs.dateFormatPattern`, except the year is always displayed with the format `yyyy`. For example, when the property `applCorePrefs.dateFormatPattern` is formatted `mm/dd/yy`, the property `applCorePrefs.longDateFormatPattern` is formatted `mm/dd/yyyy`. When the property `applCorePrefs.dateFormatPattern` is formatted `mm/dd/yyyy`, the property `applCorePrefs.longDateFormatPattern` is formatted `mm/dd/yyyy`.

Example 21-28 Formatting Dates with `dateFormatPattern`

```java
<af:column sortProperty="OrderDate" filterable="true" sortable="true" headerText="#{bindings.NlsOrders1.hints.OrderDate.label}" id="c1">
    <f:facet name="filter">
        <af:inputDate value="#{vs.filterCriteria.OrderDate}" id="id8" >
            <af:convertDateTime pattern="#{applCorePrefs.dateFormatPattern}" />
        </af:inputDate>
    </f:facet>
    <af:inputDate value="#{row.bindings.OrderDate.inputValue}" label="#{bindings.NlsOrders1.hints.OrderDate.label}" required="#{bindings.NlsOrders1.hints.OrderDate.mandatory}" shortDesc="#{bindings.NlsOrders1.hints.OrderDate.tooltip}" id="id1">
        <f:validator binding="#{row.bindings.OrderDate.validator}" />
        <af:convertDateTime pattern="#{applCorePrefs.dateFormatPattern}" />
    </af:inputDate>
</af:column>
```

21.4.1.2 Formatting Current Dates

Applications often show the current date on the user interface. They are mapped to the `java.sql.Date` data type in the entity object/view object layer. Correctly identifying the date requires the use of particular APIs.

A server date is calculated by truncating the time portion of the current time from the system clock. However, it may not be appropriate to display the server date to end users if they are not located in the server time zone. For example, when creating an order, the order form may display with the order date filled out for the end user with the current date. In this case, the order date must be the end user's local date rather than the server date. A server in the US may be serving an end user in China, whose local date may be one day ahead due to time zone differences. It is necessary to adjust the server date to the end user's local date.

The following methods return the current date in UPTZ:

- `ApplSessionUtil.getCurrentLocalDate()`: This method returns an instance of `java.util.Date`. 

Format Date and Timestamp Values

•

ApplSessionUtil.getCurrentLocalSqlDate(): This method returns an instance of
java.sql.Date.

These methods can be used in JSF pages, entity object files, and Java bean classes to
retrieve a current date in UPTZ.
Example 21-29 shows how to adjust the server date to the local date in the Java bean
using the getCurrentLocalDate() method.
Example 21-30 shows how to display the current date in the JSF page.
example shows how to set the current date in UPTZ as a value in entity object files.
The Required Date field in Figure 21-8 should default to the user's preferred time zone
rather than the server time zone, which, in this case, shows a date value with the
format yyyy.mm.dd.
Figure 21-8

Example of a Field to Default to User's Time

Example 21-29

Getting the Current Date in UPTZ Using java Bean Classes

import java.sql.Date;
import oracle.apps.fnd.applcore.common.ApplSessionUtil;
...
Date currentDate = ApplSessionUtil.getCurrentLocalSqlDate();
...

Example 21-30

Displaying the Current Date in the JSF Page

<af:outputText value="#{applCorePrefs.currentLocalSqlDate}" id="ot1">
<af:convertDateTime pattern="#{applCorePrefs.dateFormatPattern}"/>
</af:outputText>

Example 21-31

Setting the Current Date in UPTZ as a Value in Entity Object Files

<Attribute
Name="CreationDate"
IsNotNull="true"
ColumnName="CREATION_DATE"
SQLType="TIMESTAMP"
Type="java.sql.Date"
ColumnType="DATE"
TableName="FND_TERRITORIES_B"
RetrievedOnInsert="true">
<DesignTime>
<Attr Name="_DisplaySize" Value="11"/>
</DesignTime>

21-18 Oracle Fusion Applications Developer's Guide


21.4.1.3 Formatting Timestamp Values

Dates with time values (called datetimes) are mapped to java.sql.Timestamp in the entity object/view object layer. Format these fields according to the user’s date and time formatting preferences.

All datetime fields must have at least one of these three patterns in the af:convertDateTime tag to be formatted according to the user’s date and time preferences.

**Note:**

There is a single space between the two Expression Language expressions.

- **Pattern 1**
  
  pattern="#{applCorePrefs.UPTZPattern}"

  applCorePrefs.UPTZPattern returns a pattern by combining the selection of the Date Format and the Time Format in User Preferences. For example, if a user selects M.d.yyyy (4.28.2010) from the Date Format list, and HH.mm (00.55) from the Time Format list, applCorePrefs.UPTZPattern returns a pattern M.dd.yyyy HH:mm. Therefore, applCorePrefs.UPTZPattern may return a pattern with or without Seconds, depending on the selection in User Preferences.

- **Pattern 2**
  
  This pattern is applicable if it always needs to display the Seconds part of a datetime value.

  pattern="#{applCorePrefs.DateFormatPattern} #{applCorePrefs.timeFormatPatternWithSeconds}"

- **Pattern 3**
  
  pattern="#{applCorePrefs.DateFormatPattern} #{applCorePrefs.timeFormatPatternWithoutSeconds}"

  This pattern is applicable if it does not need to display the Seconds part of a datetime value.

- **Pattern 4**
  
  pattern="#{applCorePrefs.timeFormatPattern}"

  This pattern returns the value of timeFormatPattern based on user preferences.

- **Pattern 5**
  
  pattern="#{applCorePrefs.timeFormatPatternWithSeconds}"

  This pattern returns the value of getTimeFormatPattern(), appending seconds to the format if the format does not already contain seconds (hh.mm.ss).
The pattern enforces the addition of the seconds attribute, depending on the value of **timeFormatPattern**.

Examples are as follows.

If the value of **timeFormatPattern** is a **hh.mm.ss**, then **timeFormatWithSeconds** produces the format **a hh.mm.ss**.

If the value of **timeFormatPattern** is a **hh.mm**, then **timeFormatWithSeconds** produces the format **a hh.mm.ss**.

- **Pattern 6**

  ```
  pattern="#{applCorePrefs.timeFormatPatternWithoutSeconds}"
  ```

  This pattern returns the value of the method **getTimeFormatPattern()**, and removes the seconds portion from the time format. This pattern enforces the removal of seconds attributes depending on the value of **timeFormatPattern**.

  Examples are as follows.

  If the value of **timeFormatPattern** is a **hh.mm.ss**, then **timeFormatWithoutSeconds** produces the format **a hh.mm**.

  If the value of **timeFormatPattern** is a **hh.mm**, then **timeFormatWithoutSeconds** produces the format **a hh.mm**.

Both the filter for the date-time and the date-time field should be formatted according to the user's date and time formatting preferences, as shown in **Example 21-32**. If this is not done, these fields will not be processed according to the user's preferences.

The Last Saved field at the top right of Figure 21-9 is an example of how a date-time is formatted according to the user's preferences in Oracle Fusion Applications. In this case, the date and time are displayed in the format **mm.dd.yy** and the time **hh:mm AM**.

**Figure 21-9   Example of a Formatted Date and Time**

**Example 21-32   Formatting Timestamp Values**

```
21.4.2 What Happens When You Format Dates and Timestamps

When dragging and dropping a view object containing an attribute of type `java.sql.Date` or `java.sql.Timestamp`, Applications Core generates code that binds to the `dateFormatPattern` (date fields) or `UPTZPattern` (timestamp fields) property in the `applCorePrefs` managed bean.

Example 21-33 shows bindings to the `dateFormatPattern` property. The pattern according to which the date value is formatted is picked up from the Preferences bean in Applications Core, which in turn accesses these values from the LDAP server configured with users' roles and policies.

This type of code is generated for both date and timestamp fields. For date fields, you should add the `type="date"` attribute, whereas for timestamp fields, you should add the `type="both"` attribute, and make the pattern attribute equal to one of the three patterns discussed above.

Example 21-33  Bindings to the dateFormatPattern Property

```xml
<af:panelLabelAndMessage label="#{bindings.OrderDate.hints.label}" id="plam1">
    <af:outputText value="#{bindings.OrderDate.inputValue}" id="ot1">
        <af:convertDateTime pattern="#{applCorePrefs.dateFormatPattern}"/>
    </af:outputText>
</af:panelLabelAndMessage>
```

21.4.3 What Happens at Runtime: How Dates and Timestamps Are Formatted

At runtime, the bindings generated at design time are executed. Dates and Timestamps are displayed according to user preferences for date and time formatting patterns (for example, 01/01/10 and 01/01/10 01:05:00).

21.4.4 Standards and Guidelines for Formatting Dates and Timestamps

The following standards and guidelines apply to formatting dates and timestamps.

- All date-only fields must be represented only by `java.sql.Date` data types.
- When a value bound to a field is date-only of type `java.sql.Date`, do not set the time zone to `af:convertDateTime`.

21.5 Formatting Time Zones

Date values with time (called datetime values) are mapped to `java.sql.TimeStamp` in the entity object and view object layer. In Oracle Fusion Applications, datetime values can be calculated according to one of these time zones:

- **Server Reporting Time Zone**
  
  This is the default and simply is the time zone of the server's operating system.

- **User Preferred Time Zone (UPTZ)**
  
  When UPTZ is used, the date-time changes in the UI when a user changes the Time Zone preference in User Preferences. For example, if a user switches the Time Zone preference from UTC-08:00 to UTC-11:00, the date-time 04/27/2010 01:00:00 will change to 04/26/2010 23:00:00. The value in the database is not
changed, and the conversion occurs after the value is retrieved from the database and before it is shown on the user interface.

- **Legal Entity Time Zone (LETZ)**

  LETZ is defined with legal entities. It is an attribute of a legal entity business object. For example, when creating a legal entity ORACLE-CDC, it can have a time zone attribute (LETZ) which value is UTC+08:00. Thus a date-time 04/27/2010 01:00:00 in UTC will change to 04/27/2010 09:00:00 accordingly. The value in the database is not changed, and the conversion occurs after the value is retrieved from the database and before it is shown on the user interface.

The business logic of the application decides which time zone to use to calculate the date-time value.

### 21.5.1 How to Format Time Zones

The different types of tags required to implement either the User Preferred Time Zone or Legal Entity Time Zone are shown in this section.

**Before you begin:**

Create an entity object and a view object with date fields, and include the java.sql.Timestamp attribute type. Note that the middle-tier time zone and the database time zone always must match. If they do not match, the conversions will be invalid.

**Using the Server Reporting Time Zone**

Use Example 21-34 if the date-time value is to be calculated according to the Server Reporting Time Zone.

This is to say that nothing must be explicitly added into the automatically generated af:convertDateTime tag.

**Using the User Preferred Time Zone**

Use this code sample to calculate the date-time value according to the User Preferred Time Zone.

**Converting a Timestamp to the Legal Entity Time Zone**

Follow this method to convert a timestamp value to the Legal Entity Time Zone:

Retrieval of the Legal Entity time zone. Each legal entity has a LETZ. Determining a LETZ starts with a legal entity’s address that is used to look up a specific location and a unique time zone attribute. This time zone is the LETZ for the legal entity.

Related tables in the database are:

- FND_TIMEZONES_B and FND_TIMEZONES_TL, tables to record time zone related information.
- XLE_REGISTRATIONS, a table of Legal Entity data.
- HZ_LOCATIONS, a table to store geographical locations.

Use this PL/SQL API to obtain the Legal Entity Time Zone code. This API is found in the package XLE_LE_TIMEZONE_GRP and is to be called in the following way:

Get_Le_Tz_Code(?,?)

The ? indicates an input parameter.
Example 21-36 shows sample code for using Get_Le_Tz_Code(?,?)

To retrieve the current date and time in the Legal Entity Time Zone, use this PL/SQL API to calculate the current date and time in the Legal Entity Time Zone. This is particularly useful while creating or saving a transaction, when the legal entity date and time is to be displayed. This API is found in the XLE_LE_TIMEZONE_GRP package and is to be called in the following way:

Get_Le_Sysdate_Time(?,?)

The ? indicates an input parameter.

Example 21-37 uses this API to determine the current date and time in the LETZ.

This function will invoke the API (Get_DefaultLegalEntity) when the input parameter passed is the Business Unit or the Inventory Organization.

To retrieve the server date and time into the LETZ, use this PL/SQL API. This is useful while retrieving an already saved transaction and showing the already saved timestamp in the LETZ. This API is found in the XLE_LE_TIMEZONE_GRP package and is to be called in the following way:

Get_Le_Day_Time(?,?,?)

The ? indicates an input parameter.

Example 21-38 uses Get_Le_Day_Time(?,?,?) to convert a given date and time from the server time zone to the LETZ.

Here the p_trxn_date is the transaction date and time value in the server time zone.

To convert the legal entity date and time into the server time zone, use this PL/SQL API. This API is found in the XLE_LE_TIMEZONE_GRP package and is to be called in the following way:

Get_Server_Day_Time(?,?,?)

The ? indicates an input parameter.

This API will be for reverse LETZ conversion. Calling applications can pass in the Legal Entity Date Time and the Business Unit/Inventory Organization/Legal Entity ID to convert the Legal Entity datetime to the server datetime.

Example 21-39 uses Get_Server_Day_Time(?,?,?) to convert the LETZ timestamp value to server time zone.

These PL/SQL APIs must be called in the Java entity object implementation classes for time zone conversion. The JSF code corresponding to the date field need not be changed.

Here is a sample of a JSF date field in the server time zone and its corresponding conversion into the LETZ. The code in the JSF page corresponding to a date field is shown in Example 21-40.

As shown, you do not need to change the date field’s snippet in the JSF page, or add a time zone attribute in the af:convertDateTime tag for this field.

The calculation of LETZ date values should be triggered when changing the value for the Business Unit field in the user interface. The value change should invoke a method that can calculate values according to the LETZ in the EOImpl class. In this case, this method is the setOrgID() method, shown in Example 21-41.
In Figure 21-10, the field Receipt Date is an example of how a date-time or a timestamp field in Oracle Fusion Applications is calculated according to the user’s preferred time zone rather than the server time zone.

**Figure 21-10 Converting to User Preferred Time Zone**

In Figure 21-11, the Business Unit field determines the LETZ according to which the date fields Date and Terms Date on the right of the screenshot are converted in Oracle Fusion Applications. Thus, the fields Date and Terms Date change according to the Business Unit field in the Create Invoice transaction. In this case, the dates display in the format mm/dd/yy.

**Figure 21-11 Converting to Legal Entity Time Zone**

**Example 21-34 Using the Server Reporting Time Zone**

```
<af:outputText value="#{bindings.StartDate.inputValue}" ... />
<af:convertDateTime type="both" pattern="#{applCorePrefs.UPTZPattern}" />
</af:outputText>
```

**Example 21-35 Using the Preferred Time Zone**

```
<af:outputText value="#{bindings.StartDate.inputValue}" ... />
<af:convertDateTime type="both" pattern="#{applCorePrefs.UPTZPattern}"
   timeZone="#{applCorePrefs.UPTZ}" />
</af:outputText>
```

**Example 21-36 Using Get_Le_Tz_Code(?,?)**

```
l_timezone_code = XLE_LE_TIMEZONE_GRP.Get_Le_Tz_Code('BUSINESS_UNIT_ID', p_ou_id);
l_timezone_code = XLE_LE_TIMEZONE_GRP.Get_Le_Tz_Code('INVENTORY_ORG_ID', p_inv_org_id);
l_timezone_code = XLE_LE_TIMEZONE_GRP.Get_Le_Tz_Code('LEGAL_ENTITY_ID', p_le_id);
```

**Example 21-37 Determining the Current Date and Time in the LETZ**

```
l_sysdate = XLE_LE_TIMEZONE_GRP.Get_Le_SysDate_Time('BUSINESS_UNIT_ID', p_ou_id);
l_sysdate = XLE_LE_TIMEZONE_GRP.Get_Le_SysDate_Time('INVENTORY_ORG_ID', p_inv_org_id);
l_sysdate = XLE_LE_TIMEZONE_GRP.Get_Le_SysDate_Time('LEGAL_ENTITY_ID', p_le_id);
```
Example 21-38  Converting a Date and Time to the LETZ Using Get_Le_Day_Time

```java
l_le_day_time = XLE_LE_TIMEZONE_GRP . Get_Le_Day_Time ('BUSINESS_UNIT_ID', p_ou_id, p_trxn_date);
l_le_day_time = XLE_LE_TIMEZONE_GRP . Get_Le_Day_Time ('INVENTORY_ORG_ID', p_inv_org_id, p_trxn_date);
l_le_day_time = XLE_LE_TIMEZONE_GRP . Get_Le_Day_Time ('LEGAL_ENTITY_ID', p_le_id, p_trxn_date);
```

Example 21-39  Converting the LETZ Timestamp to the Server Time Zone with Get_Server_Day_Time(?,?,?)

```java
l_Server_Date := XLE_LE_TIMEZONE_GRP . Get_Server_Day_Time ('INVENTORY_ORG_ID', p_inv_org_id, p_le_Date);
l_Server_Date := XLE_LE_TIMEZONE_GRP . Get_Server_Day_Time ('BUSINESS_UNIT_ID', p_ou_id, p_le_Date);
l_Server_Date := XLE_LE_TIMEZONE_GRP . Get_Server_Day_Time ('LEGAL_ENTITY_ID', p_inv_org_id, p_le_Date);
```

Example 21-40  Converting a Date Field into the LETZ

```java
<af:inputDate value="#{bindings.InvoiceDate.inputValue}" label="#{bindings.InvoiceDate.hints.label}" required="#{bindings.InvoiceDate.hints.mandatory}" shortDesc="#{bindings.InvoiceDate.hints.tooltip}" showRequired="true" autoSubmit="true" valueChangeListener="#{pageFlowScope.invoiceActionsBean.invoiceDateValueChangeListener}" columns="#{bindings.InvoiceDate.hints.displayWidth}" disabled="#{pageFlowScope.matchPopupVisible}" id="id2">
  <f:validator binding="#{bindings.InvoiceDate.validator}"/>
  <af:convertDateTime pattern="#{applCorePrefs.dateFormatPattern}"/>
</af:inputDate>
```

Example 21-41  Using the setOrgID() Method

```java
public void setOrgID(Long orgID)
{

  // To convert the current timestamp to the LETZ, call another method
  this.setInvoiceDate(this.getLEDateTime(orgID, this.getDBTransaction()));
}
```

```
//This method calls the PL/SQL API to get the current date and time in the LETZ.
public Date getLEDateTime(Long orgId, DBTransaction dbTransaction)
{

  // 1. Define the PL/SQL block for the statement to invoke
  String stmt = "begin ? := XLE_LE_TIMEZONE_GRP.Get_Le_Sysdate_Time(?,?);  end;";
  // 2. Create the CallableStatement for the PL/SQL block
  st = (OracleCallableStatement)dbTransaction.createCallableStatement(stmt, 0);
  // 3. Register the positions and types of the OUT parameters
  st.registerOutParameter(1, Types.DATE);
  // 4. Set the bind values of the IN parameters
  st.setObject(2,"BUSINESS_UNIT_ID");
  st.setObject(3,orgId);
  // 5. Execute Query
  st.executeUpdate();
  return st.getDate(1);
}
```
21.5.2 How to Format Time with and without Seconds

The user preferred pattern for time formatting might include seconds. In some cases, it may be necessary for an application to display a time value with seconds under one set of circumstances, and without seconds under another.

By default, the display of seconds depends on the user preferred time format. Example 21-42 prints a datetime value in the user preferred format.

Example 21-43 prints a datetime value in the user preferred format with seconds always included.

Example 21-44 prints a datetime value in the user preferred format without seconds.

Example 21-42  Formatting in the User Preferred Datetime Pattern

Example 21-43  Formatting in the User Preferred Datetime Pattern with Seconds

Example 21-44  Formatting a Datetime Value in the User Preferred Datetime Pattern without Seconds

21.5.3 How to Format Invariant Time Zone Values

Some date-time values are not associated with a specific time zone. For example, an application may execute a job at 9 AM local time in every location across different time zones. Such values are called invariant or floating times. To print an invariant time zone value, use the default time zone such that no specific time zone is applied to the value.

When printing a date-time value for a specific time zone derived from an invariant time zone value, you may need to adjust the formatting so as to neutralize the effect of time zone conversion. This is because the server default time zone is applied implicitly.

21.5.4 What Happens When You Format Time Zones

When you drag and drop a timestamp field onto the JSF page, Applications Core generates the tags shown in Example 21-45.

You need to change the tags depending on whether the field is to be calculated according to the UPTZ or the server time zone (invariant timestamp values), for example Default : Server Time Zone.

This is necessary only for invariant timestamp values. Invariant timestamp values are timestamps in which varying time zone will not make a difference to the business logic of the application.

Example 21-47 shows tags with a user preferred time zone.

At design time, Applications Core uses the date-time sensitive custom property to generate bindings to the time zone attribute on the Oracle ADF Faces Date Time Converter. The attribute is bound to the applCorePrefs managed bean.
21.5.5 What Happens at Runtime: How Time Zones Are Formatted

At runtime, the Applications Core managed bean `applCorePrefs`, implemented by `oracle.apps.fnd.applecore.common.PreferencesBean` and registered with the `faces-config.xml` file, retrieves the relevant formatting masks from Applications Session.

By default, date-time data may display as shown in Example 21-48.

**Example 21-48  Date Time Data Format**

1/1/2009 12:34 AM        for the pattern "M/d/yyyy hh:mm a"

21.5.6 Standards and Guidelines

The following standards and guidelines apply to formatting time zones:

- All date time fields must be represented only by `java.sql.Timestamp` data types (used by default in time zone view object attributes).
- The database and middle-tier Timezone must always be the same in Oracle Fusion Applications.

21.6 Formatting Numbers, Currency and Dates Using Localization Expression Language Functions

Expression Language functions provide an alternative to the formatting procedures described in Formatting Currency, Formatting Numbers and Formatting Time Zones.

21.6.1 How to Format Numbers, Currency and Dates Using Expression Language Functions

Oracle ADF Faces Expression Language functions of the type `af:formatNamed` and `af:format` only support `String` objects as parameters. Consequently, other object types such as `Date` and `BigDecimal` must be converted to the `String` object type.

For example, when binding the date object `dateValue` as shown in Example 21-49, the `dateValue` object must be converted to a `String` object by calling the `toString()` method.
However, the `toString()` method does not support Oracle Fusion Applications user preferences. Oracle Fusion Applications thus require the use of Expression Language format functions to convert the following data objects to String objects:

- Number and currency objects:
  - `java.math.BigDecimal`
  - `java.lang.Integer`
  - `java.lang.Long`

- Date and DateTime (Timestamp) objects:
  - `java.sql.Date`
  - `java.sql.Timestamp`

You can format numbers, currency and dates using Expression Language functions.

**Example 21-49  Binding a Date Object**

```af
af:formatNamed(bundle.NOTE_MESSAGE, 'BIRTHDAY', dateValue)
```

### 21.6.1.1 Formatting Numbers Using Expression Language Functions

Use the following Expression Language functions to format numbers.

The number Expression Language formatting function is shown in **Example 21-50**.

Returns the formatted number value using the user preferences for the number format mask, grouping separator and decimal separator.

This function produces the tag shown in **Example 21-51**.

An additional Expression Language formatting function for numbers is shown in **Example 21-52**.

Returns the formatted number value using the user preferences for the number format mask, grouping separator and decimal separator.

Overrides the scale—the number of digits following the decimal point—of the user preferred number format pattern using the value assigned to `maxFractionDigit`.

This function produces the tag shown in **Example 21-53**.

**Example 21-50  formatNumber(java.lang.Number value) Function**

```fnd
fnd:formatNumber(java.lang.Number value)
```

**Example 21-51  Tag Produced by the Function fnd:formatNumber(java.lang.Number value)**

```af
<af:convertNumber pattern="#{applCorePrefs.numberFormatPattern}"/>
```

**Example 21-52  formatNumber2(java.lang.Number value, int maxFractionDigit) Function**

```fnd
fnd:formatNumber2(java.lang.Number value, int maxFractionDigit)
```

**Example 21-53  Tag Produced by the Function fnd:formatNumber2(java.lang.Number value, int maxFractionDigit)**

```af
<af:convertNumber pattern="#{applCorePrefs.numberFormatPattern}" maxFractionDigits="your scale here"/>
```
21.6.1.2 Formatting Currency Using Expression Language Functions

Use the Expression Language function shown in Example 21-54 to format currency.

Returns the formatted currency amount value in numeric form along with the relevant currency code. Applications Core uses the currency code as defined in FND_CURRENCIES to format the currencyAmount value, rather than the number format mask preference. User preferences for grouping and decimal separators are used to format the value.

This function produces the tag shown in Example 21-55.

**Example 21-54**  
`fnd:formatCurrency(java.lang.Number currencyAmount, java.lang.String currencyCode)` Function

```
fnd:formatCurrency(java.lang.Number currencyAmount, java.lang.String currencyCode)
```

**Example 21-55**  
Tag Produced by the Function  
`fnd:formatCurrency(java.lang.Number currencyAmount, java.lang.String currencyCode)`

```
<af:convertNumber type="currency"  
currencyCode="#{bindings.quantityCurrencyCode.inputValue}"  
pattern="#{fnd:currencyPattern(bindings.quantityCurrencyCode.inputValue)}"/>
```

21.6.1.3 Formatting Dates Using Expression Language Functions

Use the Expression Language function shown in Example 21-56 to format dates.

Returns the formatted date value based on the user preferred date format mask.

This function produces the tag shown in Example 21-57.

Use the Expression Language function shown in Example 21-58 to format date time values.

Returns the formatted date time value using the user preferences for the date and time format masks and time zone.

This function produces the following tag as shown in Example 21-59.

Use the Expression Language function shown in Example 21-60 to format date time values with user formatting masks and the user-specified time zone.

Returns the formatted date time value using the user preferences for date and time format masks and the user-specified time zone.

This function produces the tag shown in Example 21-61.

**Example 21-56**  
`fnd:formatDate(java.util.Date dateValue)` Function

```
fnd:formatDate(java.util.Date dateValue)
```

**Example 21-57**  
Tag Produced by the Function `fnd:formatDate(java.util.Date dateValue)`

```
<af:convertDateTime pattern="#{applCorePrefs.dateFormatPattern}"/>
```

**Example 21-58**  
`fnd:formatDateTime(java.util.Date dateTimeValue)` Function

```
fnd:formatDateTime(java.util.Date dateTimeValue)
```
Example 21-59  Tag Produced by the Function fnd:formatDateTime(java.util.Date dateTimeValue)

<af:convertDateTime type="both" timeZone="#{applCorePrefs.UPTZ}" pattern="#{applCorePrefs.UPTZPattern}"/>

Example 21-60  fnd:formatDateTimeTZ(java.util.Date dateTimeValue, java.util.TimeZone timeZone) Function

fnd:formatDateTimeTZ(java.util.Date dateTimeValue, java.util.TimeZone timeZone)

Example 21-61  Tag Produced by the Function fnd:formatDate(java.util.Date dateTimeValue, java.util.TimeZone timeZone)

<af:convertDateTime type="both" timeZone="<your timezone>" pattern="#{applCorePrefs.UPTZPattern}"/>

21.6.2 What Happens When You Format Numbers, Currency and Dates Using Expression Language Functions

Applications Core formats the value as defined by the Expression Language function and produces the tags described in this section.

For example, the date formatting Expression Language function produces a tag such as the one shown in Example 21-62.

Example 21-62  Tag Produced by Expression Language Date Formatting Function

<af:convertDateTime pattern="#{applCorePrefs.dateFormatPattern}"/>

21.6.3 What Happens at Runtime: How Currency, Dates and Numbers and Time Zones are Formatted Using Expression Language Functions

For more information about what happens at runtime when you format numbers, currency and dates using Expression Language functions, see Formatting Currency, Formatting Numbers and Formatting Time Zones.

21.7 Implementing Bi-directional Support

Oracle Fusion applications should be enabled to provide bi-directional support. Examples of bi-directional languages are Arabic (Middle East and North Africa) and Hebrew (Israel). These scripts are generally written from right to left. The entire UI must also render from right to left. That is, if you took a standard English UI and placed a mirror next to your computer screen, the image in the mirror is exactly what is expected of speakers of bi-directional languages. Some rules come into play, such as how to correctly render English text or numerals embedded in a bi-directional script, but this generally is handled by underlying technologies (in this case, Oracle ADF). An example of this is that the following text, which translates to United Arab Emirates:

Oracle

for a user with English locale must be displayed as:

الإمارات العربية المتحدة

for a user with Arabic Locale.
21.7.1 How to Implement Bi-directional Support

Before you begin, create an entity and a view object from ADF Business Components. Drag and drop some of the attributes from the view object into a JSF page.

21.7.1.1 Making Panels and Columns Provide Bi-directional Support

UI Components such as af:panelGroupLayout and af:column have an align or an halign attribute that allows developers to align a particular component in one direction. A typical example of this is a number field that is always aligned to the right by Oracle Fusion Applications standards. Now, if a UI component (a non-number field) is hardcode-aligned to the left or right in the JSF page, the implication in the Arabic locale is that the UI component does not switch to its corresponding mirror image position. Consequently, there are discrepancies in the user interface in the Arabic locale.

To correct this situation, you must not hardcode a UI component (non-number field) to the left or right. You should always use the start or end values for the halign and align attributes to make them bi-directional compatible, as shown in Example 21-63.

Here, the halign attribute for the panel group has been set to end, which means that this component will be bi-directional compatible and will switch to its mirror image position in the Arabic locale.

In the English locale shown in Figure 21-12, the panel containing the three buttons Advanced, Search and Reset is aligned to the right.

**Figure 21-12** Showing Right-aligned Buttons for English Locale

![Right-aligned Buttons for English Locale](image)

Therefore, the expected behavior is that, in the Arabic locale shown in Figure 21-13, the panel, with the buttons’ text in Arabic, is aligned to the left.

**Figure 21-13** Showing Left-aligned Buttons for Arabic Locale

![Left-aligned Buttons for Arabic Locale](image)

**Example 21-63** Making align Attributes Bi-directional

```xml
<af:panelGroupLayout id="pg15" layout="horizontal" halign="end">
    <af:commandButton text="#{ResourcesGenBundle['Action.Advanced.AdvancedSearch']}
                      actionListener="#{ItemRelationshipRegionalSearchBean.
                      onAdvancedButtonClick()"
                      id="cb4"/>
    <af:spacer width="8" height="10" id="s7"/>
    <af:image source="/images/seperator_img.png" id="i2" shortDesc=""/>
    <af:spacer width="8" height="10" id="s8"/>
</af:panelGroupLayout>
```
21.7.1.2 Making Images Provide Bi-directional Support

In Oracle Fusion applications, all UI components should provide bi-directional support. This means that, in bi-directional locales such as Arabic and Hebrew, UI components should be shown from right-to-left instead of left-to-right. This also includes images that are shown on the UI to give a pictorial representation of an instruction to the end user. If an image is directional (unsymmetrical), it should be flipped and shown as its corresponding mirror image from right-to-left. However, if an image is non-directional (symmetrical) it need not be flipped since its mirror image is the image itself. To make an image bi-directional, you have to use the function fnd:bidiImage(<Image-Path>), shown in Example 21-64, that is provided as a part of the ApplCore library.

In bi-directional locales, instead of displaying the image search_ena.png directly, the fnd:bidiImage suffixes an _rtl just before the start of the extension to the parameter file-name given to it (in this case, the image icon search_ena.png becomes search_rtl_ena.png), and then displays the image search_rtl_ena.png on the JSF page. Now, if the latter image has been stored as a mirror image of the former, bi-directionality is ensured for the image.

Figure 21-14 shows how the search image next to the Search by Year option appears in an English locale. Specifically, the icon displays an arrow pointing to the right to the right of the Search by Year field.

Figure 21-14    Showing Command Image Link in English Locale

Figure 21-15 shows how the command image link supports bi-directional behavior in an Arabic locale. Specifically, the icon displays an arrow pointing to the left to the left of the Search by Year field.
21.8 Supporting Mnemonic Keys

A mnemonic key, also called a soft key, is a shorthand name for a key, command, or menu option. Mnemonic keys are used in Oracle Fusion applications to activate the buttons in a page. This is made possible by using a $<\text{Modifier}>+<\text{Access Key}>$. Note that modifiers are browser-specific. Internet Explorer, Safari and Google Chrome use the $\text{Alt}$ key; Firefox uses $\text{Alt+Shift}$.

The Access Key is defined in the source code by the developer.

In Oracle Fusion applications, the access key for a particular button must be externalized so that the keys for different locales are different. Do not hardcode the $\text{accessKey}$ or the $\text{textAndAccessKey}$ attribute provided as a child attribute of the $\text{af:commandButton}$ tag. Hard-coding these keys would cause mnemonic key issues. For example, if an English letter is hardcoded as an access key in the JSF layer, an Arabic user may not be able to use this key because there are no English letters on his keyboard. Instead the $\text{accessKey}$ and $\text{textAccessKey}$ attributes should be externalized using either the $\text{applCoreBundle}$ or any resource bundle defined in an Oracle Fusion application's project.

21.8.1 How to Implement Mnemonic Key Support

Before you begin, create an entity object and a view object from ADF Business Components. Drag and drop some of the attributes from the view object into a JSF page. Create a panel group and add a command button to it.

There are two attributes provided by Oracle ADF for mnemonic keys for buttons: $\text{accessKey}$ and $\text{textAndAccessKey}$. The attribute $\text{accessKey}$ takes as a value just the access key to be typed by the user, whereas the attribute $\text{textAndAccessKey}$ takes as a value the text to be displayed on the button and the short-hand key to be typed by the user. You must make sure that the $\text{accessKey}$ and $\text{textAndAccessKey}$ attributes are never hard-coded. Both should be externalized and read from resource bundles. In the case of the $\text{textAndAccessKey}$ attribute, the value of the access key usually is preceded by an ampersand ($\&$) in the resource bundle. In case the ampersand sign is not present in the bundle, the first letter of the text is treated as the access key.
How to use the `textAndAccessKey` attribute using the `applCoreBundle` is shown in Example 21-65.

How to use the `textAndAccessKey` attribute using the resource bundles is shown in Example 21-66.

Here, the resource bundle `TemporaryBundle` must be defined in the JSF page containing the `af:commandButton` tag. Also, the `textAndAccessKey` has been externalized and is read from the `applCore` bundle and the resource bundle, as shown in the examples, which is the correct way to define mnemonic keys. The values for this particular id `SAVE` in the resource bundles are "Save" in the English locale, "\uC800\uC7A5(&S)" in the Korean locale (S is the access key) and "\u062D\u0641\& \u0638" (\u0638 is the access key) in the Arabic locale.

Example 21-67 shows how to externalize the `accessKey` attribute using the `applCoreBundle`.

Example 21-68 shows how to externalize the `accessKey` attribute using a resource bundle.

Here, the `accessKey` attribute has been externalized and is read from the Applications Core bundle and the resource bundle as shown in the examples. Note that the `TemporaryBundle` has to be defined as the `viewControllerBundle` in the JSF page containing the command button. The values for this particular resource ID `ACCESSKEY` are "S" in the English locale, ""\uC800" in the Korean locale and "\u0638" in the Arabic locale, as defined in the resource bundle.

**Note:**

- When a `textAndAccessKey` attribute has a value with more than one "&", the letter following the first "&" is chosen as the access key.
- When an `accessKey` attribute has more than one letter, the first letter is chosen as the access key.

In Figure 21-16, the buttons on the top right (Run Validations, View Exceptions, Open Target Period) are examples of buttons with soft keys that must not be hardcoded, so that they can change for different locales.

**Figure 21-16   Example of Buttons with Soft Keys**

Example 21-65 Using the `textAndAccessKey` Attribute Using the `applCoreBundle`

```xml
<f:facet name="actionButtonBar">
  <af:panelGroupLayout layout="horizontal"
    id="panelGroupLayout11"
    styleClass="AFStretchWidth">
```
Example 21-66  Using the textAndAccessKey Attribute Using Resource Bundles

Example 21-67  Externalizing the accessKey Attribute Using the applCoreBundle

Example 21-68  Externalizing the accessKey Attribute Using a Resource Bundle

21.9 Implementing Localization Formatting in ADF Desktop Integration

With ADF Desktop Integration technology, users can browse, edit and upload data through a Microsoft Excel spreadsheet. Localization formatting differs from ADF Faces pages:

- Number and date/time values are formatted based on the format pattern specified by the styles in the Excel spreadsheets. The format pattern follows Microsoft standard instead of Java standard.
- Number separators are decided by the client operating system locale.

In ADF Faces pages, number and date/time values are formatted based on the format patterns selected in the Oracle Fusion application’s Preference UI. Number separators are extracted from the application’s number format patterns.

21.9.1 How to Format Numbers

Number separators in ADF Desktop Integration spreadsheets vary according to the client OS locale. Number formatting is decided by the style defined in Excel spreadsheet. For example, when opening an ADF Desktop Integration spreadsheet with French Windows XP and Excel, number 1000.5 (One thousand point 5) displays as 1 000,5. It is displayed as 1.000,5 in German Windows XP and Excel. Assuming one fractional digit is specified in the style.

To achieve this, developers need to specify correct styles for numbers at design time.

21.9.1.1 Formatting Numbers

Before you begin, create a workable ADF Desktop Integration spreadsheet and have the attributes in place. Related attributes are of types: java.math.BigDecimal, java.lang.Integer, or java.lang.Long.

Define styles for each type of cells and configure the cell appearance using the Oracle Fusion Applications UI standard.

Decimal Number Formatting
Follow these steps to modify the existing style for formatting decimal numbers and apply the style to the cell containing a decimal number value.

- In Excel, select Home > Cell Styles.
- Right-click the style name and select Modify from the context menu.
- On the Style dialog, click Format.
- Select the Number tab and select the Number category. Specify the number of decimal places and the format of negative numbers according to your business logic. If you want to see a grouping separator in the number values, select Use 1000 Separator.
- Click OK and save the Excel file.

**Integer Formatting**

Integer formatting is the same as decimal formatting except no fractional digit should display. This can be done by specifying 0 for Decimal places when modifying the style.

**ID Formatting**

An ID value is a sequence of digits, such as an Invoice Number, Service Request ID, or Bank Account. Displaying ID values should not vary based on a user’s preference or client OS locale. That is, ID values should be printed as text instead of numbers.

In Oracle Fusion applications, the length of an ID value typically is 13 or 18. By default, Excel formats such long numbers with Scientific Notation format, which is not desired for Oracle Fusion applications. For instance, an Invoice Number 1234567890123 (13 digits) is displayed as 1.23457E+12. It is not a valid Invoice Number when it is printed out.

To avoid the default formatting of Excel in such cases, developers need to customize the styles for ID values.

- In Excel, select Home > Cell Styles
- Right-click the style name and select Modify from the context menu.
- Click Format.
- Select the Number tab and select the Text category.
- Click OK. In the Style dialog, Figure 21-17 shows that @ is defined as the Number style.

![Figure 21-17 Selecting the Number Style](image)

- Click OK and save the file.

*Figure 21-18* shows that an ID value is displayed as text without any number formatting.
21.9.1.2 What Happens When You Format Numbers

When dragging and dropping a binding to generate the components in an ADF Desktop Integration spreadsheet, the framework generates default styles for those values based on the Java type of the values. However, these styles may not be correct for formatting number values. Therefore, you may need to refine the styles according to business logic.

21.9.1.3 What Happens at Runtime: How Numbers Are Formatted

At runtime, ADF Desktop Integration will pass the style specified in the ADF Desktop Integration unpublished spreadsheet (known as the design time file) to the published spreadsheet during the publication process. When a user opens the published spreadsheet, Excel formats the numbers according to the combination of styles and client OS locale.

21.9.2 How to Format Currency Values

Grouping and decimal separators of currency values vary based on the client OS locale, while the number of fraction digits is controlled by the style defined in the Excel spreadsheet.

You should not display currency symbols or currency codes with currency numbers in the same cell. Instead, you need to display currency symbols or currency codes in a separate cell.

21.9.2.1 Formatting Currency Values

Before you begin, create a workable ADF Desktop Integration spreadsheet and have the attributes in place. The related attribute is of type java.math.BigDecimal.

Define styles for each type of cells and configure the cell appearance using the Oracle Fusion Applications UI standard.

Follow these steps to modify an existing style for formatting currency values and apply it to the corresponding cells.

- In Excel, select **Home > Cell Styles**, right-click the style name and select **Modify** from the context menu.
- Click **Format**.
- Select the Number tab and select the Currency category. Specify the number of Decimal places and the format of negative numbers according to your business logic.
- In Oracle Fusion Applications, display the currency symbol in a separate cell. From the Symbol dropdown list, select **None**.
- Click **OK** and save the file.
21.9.2.2 What Happens When You Format Currencies

When dragging and dropping a binding to generate the components in an ADF Desktop Integration spreadsheet, the framework generates default styles for those values based on the Java type of the values. However, these styles may not be correct for formatting number values. Therefore, you may need to refine the styles according to business logic.

21.9.2.3 What Happens at Runtime: How Currency Values Are Formatted

At runtime, ADF Desktop Integration will pass the style specified in the unpublished spreadsheet (the design time file) to the published spreadsheet during the publication process. When a user opens the published spreadsheet, Excel formats the numbers according to the combination of styles and the client OS locale.

The currency code should be displayed in a cell other than the cell containing the Currency Value. The currency code in this cell should be printed as text.

21.9.3 How to Format Dates and Timestamp Values

The format of date and timestamp values in ADF Desktop Integration spreadsheets varies according to the client OS locale.

For example, when opening an ADF Desktop Integration spreadsheet with French Windows XP and Excel, the date May 25th, 2011 is displayed as 25/05/2011. It displays as 2011-05-25 in Korean Windows XP and Excel.

To achieve this, developers need to specify correct styles for date and timestamp values at design time.

21.9.3.1 Formatting Date and Timestamp Values

Before you begin, create a workable ADF Desktop Integration spreadsheet and have the attributes in place. Related attributes are of types: java.sql.Date or java.sql.Timestamp.

Define styles for each type of cell and configure the cell appearance using Oracle Fusion Applications UI standards.

**Formatting the Date**

Follow these steps to modify an existing style for formatting dates and apply it to the correspondent cells.

- In Excel, select Home > Cell Styles
- Right-click the style name and select Modify from the context menu.
- Click Format.
- Select the Number tab and select the Date category. Set the value of Locale (location) to your Windows OS locale. For example, if you are using English Windows XP, this Locale should be set to English (U.S.).
- Select a type beginning with an asterisk (*), as shown in Figure 21-19. There are two such types: one is a short date format, and the other is a long date format. Choose one according to your business requirement.
Figure 21-19  Selecting the Date Type

Note that date formats display date and time serial numbers as date values. Date formats that begin with an asterisk (*) respond to changes in regional date and time settings that are specified for the operating system. Formats without an asterisk are not affected by operating system settings.

- Click OK and save the file.

Formatting the Timestamp

Follow these steps to define a style for formatting the timestamp and apply it to the corresponding cells.

- In Excel, select Home > Cell Styles.
- Right-click the style name and select Modify from the context menu.
- Click Format.
- Select the Number tab and select the Time category. Set the value of Locale (location) to your Windows OS locale. For example, if you are using English Windows XP, this Locale should be set to English (U.S.).
- Select the type beginning with an asterisk (*), as shown in Figure 21-20.
• Click OK and save the file.

Note: With current ADF Desktop Integration support, it is impossible to display both the date and the time part of a timestamp value in a single cell if you want to see its format varying according to client OS locale.

Workaround
To display the date and time, formatted according to the client OS locale, you can use two cells: One for displaying the date part and another for displaying the time part of the timestamp. Then format each of these cells separately as described for date and time format.

21.9.3.2 Honoring Time Zones

There are two time zones involved in time zone conversion for timestamp values in ADF Desktop Integration spreadsheets. One is the time zone of the Oracle Fusion Applications server, called the server time zone. The other is the time zone of the client system on which the ADF Desktop Integration spreadsheet is open, which is called the client time zone.

At runtime, conversion between server time zone and client time zone happens automatically for timestamp values in ADF Desktop Integration spreadsheets.

For example, if the server time zone is (GMT-8:00) Pacific Time (US & Canada) and the client time zone is (GMT+8:00) Beijing, Chongqing, Hong Kong, Urumqi, the timestamp value in the database 2011-05-22 20:00:00 is converted to 2011-05-23 12:00:00 at runtime and displayed as 2011-05-23 or 12:00:00 in the ADF Desktop Integration spreadsheet, depending on the format style defined for the corresponding cells.

21.9.3.3 What Happens When You Format the Date and Timestamp

When dragging and dropping a binding to generate the components in an ADF Desktop Integration spreadsheet, the framework generates default styles for those values based on the Java type of the values. However, these styles may not be correct for formatting number values. Therefore, you may need to refine the styles according to business logic.
21.9.3.4 What Happens at Runtime: How Date and Timestamp Are Formatted

At runtime, ADF Desktop Integration will pass the style specified in the unpublished spreadsheet (the design time file) to the published spreadsheet during the publication process. When a user opens the published spreadsheet, Excel formats the date or timestamp values according to the combination of styles and the client OS locale.

Related Links

The following document provides additional information related to subjects discussed in this section:

- For more details about Excel styles, refer to the Oracle Fusion Middleware Developing Applications with Oracle ADF Desktop Integration.
- For timestamp values, time zone conversion also happens at runtime. See Honoring Time Zones for more details.

21.10 Implement Localization Formatting in Oracle BI Publisher Reports

Oracle BI Publisher can pass Oracle Fusion applications Preferences Number Format, Date Format and Time Format to Oracle BI Publisher reports using function format-date, format-number or format-currency, if Oracle BI Publisher is running in Oracle BI Publisher and Oracle Fusion applications integration mode.

Note:

To modify RTF templates, Oracle BI Publisher Template Builder for Word should be installed on Microsoft Word 2003, 2007 or higher version. After it is installed, there is a Oracle BI Publisher menu in the Word menu bar.

21.10.1 How to Format Numbers in a Oracle BI Publisher Report

To format numbers in an RTF template with an Oracle Fusion application user-preferred number format pattern, Oracle’s format-number function must be used with 'XDODEFNUM' used as a format mask, for example:

<?format-number:fieldName; 'XDODEFNUM'?>

Follow these steps to format numbers with the Oracle Fusion application user-preferred number format pattern:

- Open your RTF template file with Word.
- Select BI Publisher > Sample XML and select a sample XML file that contains sample data for the number fields to be formatted in the RTF template.
- Double-click the number field to be modified (such as ORDER_TOTAL). In the Oracle BI Publisher Properties dialog, select the Properties tab and make sure that Regular Text is selected for Type.
- Select the Advanced tab and enter the format code, as shown in Figure 21-21.
Follow these steps to format a number value in graphs in Oracle BI Publisher reports:

- To set format masks for graphs, an Oracle Fusion application Oracle BI Publisher report designer needs to change the RTF template graph code manually.
- Edit the graph and change the graph definition code in the Advanced tab of the Graph dialog in the template builder.
- Use an XSLT format such as:

```
<xsl:value-of
    select="xdoslt:xdo_format_number($_XDOXSLTCTX, current-group() / SGT_NUMBER,'XDODEFNUM')" />
```

To format a currency value on the graph.

At runtime, XDODEFNUM is replaced with the number format pattern selected in the Oracle Fusion application Preferences UI. Consequently, the number is formatted with the Oracle Fusion application number format pattern. For example, if a user selects -1'234,567 as the Number Format in the Preferences UI, as shown in Figure 21-22, the number 1000.8888 (one thousand point eight eight eight eight) is displayed as 1'000,889 in the Oracle BI Publisher report.
When using XDODEFNUM as the format mask in format-number, the format of the number is fully controlled by the Oracle Fusion application Number Format, including grouping separator, decimal separator, number of fractional digit, and form of negative numbers.

In the Oracle BI Publisher and Oracle Fusion application integration environment, if format-number is used to format a number, whether XDODEFNUM is used as the format mask, the grouping separator and decimal separator are always derived from the Oracle Fusion application Number Format.

Therefore, an integer could be formatted as:

```xml
<?format-number:fieldName; '999G999'?>
```

In this case, if the Number Format is -1'234,567, the integer 10000 is displayed as 10'000 in the Oracle BI Publisher report.

If you do not wish to display a number with fractional digits or grouping separators (such as PO numbers, Bank Accounts, or Invoice Numbers), enter `<?fieldName?>` in step 4 of [To format numbers]. In such cases, the Number Format has no effect on the field; it always is displayed as text.

### 21.10.2 How to Format Currency Values in Oracle BI Publisher

Currency fields in RTF templates can be formatted with the format-currency function:

```xml
<?format-currency:Amount_Field;CurrencyCode;displaySymbolOrNot?>
```

where

- **Amount_Field** takes the tag name of the XML element that holds the amount value in your data.

- **CurrencyCode** can be set to a static value or it can be set dynamically. If the value will be static for the report, enter the ISO three-letter currency code in single-quotes, such as 'USD'.

To set the value dynamically, enter the tag name of the XML element that holds the ISO currency code. An element containing the currency code must be present in the data.

- At runtime, the **Amount_Field** will be formatted according to the format you set up for the currency code in the report properties.
• **displaySymbolOrNot** takes as a value either 'true' or 'false' in single quotes. When set to 'true', the currency symbol will be displayed in the report based on the value for **CurrencyCode**. If you do not wish the currency symbol to be displayed, you can either enter 'false' or simply omit the parameter.

To use the format-currency function, the currency values in your data source must be in a raw format, with no formatting applied (for example: 1000.00). If the value has been formatted for European countries (for example: 1.000,00), the format will not work.

To format your currency values with the currency code passed dynamically from a particular business flow, you must have the currency code defined in your data source.

In the Oracle Fusion application and Oracle BI Publisher integration environment, Currency is passed to Oracle BI Publisher through XDODEFCC. For example, if US Dollar is selected for Currency in Preferences, as shown in Figure 21-23, at runtime, XDODEFCC is replaced with USD.

**Figure 21-23  Selecting a Currency in Oracle Fusion Application Preferences**

![Figure 21-23](image)

Therefore, to format a currency value with the Currency selected in Oracle Fusion application Preferences, you can use:

```xml
<?format-currency:Amount_Field; 'XDODEFCC'?>
```

or

```xml
<?format-currency:Amount_Field; 'XDODEFCC';'true'?>
```

To format currency values with the currency code selected in a particular business flow instead of the Currency code selected in Oracle Fusion application Preferences, you can use:

```xml
<?format-currency:Amount_Field; CurrencyCode_Field?>
```

or

```xml
<?format-currency:Amount_Field; CurrencyCode_Field;'true'?>
```
where **CurrencyCode_Field** is the tag name of the XML element that holds the currency code selected in the business flow. At runtime, **CurrencyCode_Field** is replaced with a currency code, such as USD, JPY, or EUR.

Note: Whether the currency code is set to a static value, such as XDODEFCC, or a dynamic value passed from a particular business flow, the grouping separator and decimal separator in currency values are always derived from the Number Format selected in Oracle Fusion application Preferences. The parameter currency code in the format-currency function controls the number of fraction digits and the placement of the grouping separator in the output.

**Example**

Assume that `<format-currency:Amount_Field; CurrencyCode_Field?>` is applied in the RTF template for a currency field.

In the report properties, if the format mask for USD is 9G999D99, for JPY it is 9G999 and for INR it is 9G99G99G999D99, and if a user has set the Number Format as -1,234.567 in Preferences (uses comma (,) as the grouping separator and dot (.) as decimal separator), the currency value of 1234567.89 will display as shown here, depending on the currency passed to CurrencyCode field at runtime.

- **If 'USD' is passed to CurrencyCode_Field at runtime, the value displayed is 1,234,567.89.**
- **If 'JPY' is passed to CurrencyCode_Field at runtime, the value displayed is 1,234,568.**
- **If 'INR' is passed to CurrencyCode_Field at runtime, the value displayed is 12,34,567.89.**

If a European user changes the Number Format to -1'234,567 in Preferences (uses the apostrophe (') as the grouping separator and comma (,) as the decimal separator), the currency value of 1234567.89 will display as:

- **If 'USD' is passed to CurrencyCode_Field at runtime, the value displayed is 1'234'567,89.**
- **If 'JPY' is passed to CurrencyCode_Field at runtime, the value displayed is 1'234'568.**
- **If 'INR' is passed to CurrencyCode_Field at runtime, the value displayed is 12'34'567,89.**

Follow these steps to format a currency field in the Oracle BI Publisher RTF template.

- Open the RTF template file with Word.
- Expand **BI Publisher > Sample XML** and select a sample XML file that contains sample data for the number fields to be formatted in the RTF template.
- Double-click the currency field to be modified (for instance ORDER_TOTAL). In the Oracle BI Publisher Properties dialog, click the Properties tab and make sure **Type** is set to **Regular Text**.
- Click the Advanced tab and enter the format code, for example `<format-currency:ORDER_TOTAL; 'XDODEFCC'?>`, as shown in Figure 21-24.
Follow these steps to format a currency value in graphs in Oracle BI Publisher reports:

- To set format masks for graphs, an Oracle Fusion application Oracle BI Publisher report designer needs to change the RTF template graph code manually.
- Edit the graph and change the graph definition code in the Advanced tab of the Graph dialog in template builder.
- Use an XSLT format, such as `<xsl:value-of select="xdoxslt:xdo_format_currency($XDOXSLTCTX,current-group()/'SGT_CURRENCY','XDODEFCC')" />` to format a currency value on the graph.

### 21.10.3 How to Format Dates and Timestamps in Oracle BI Publisher

In the Oracle BI Publisher RTF template, a date or timestamp can be formatted as:

```xml
<?format-date:date_string; 'ABSTRACT_FORMAT_MASK';'TIMEZONE'?>
```

where:

- **ABSTRACT_FORMAT_MASK** can be any date format pattern supported by Oracle BI Publisher, such as YYYY-MM-DD or DD/MM/YYYY.
- **TIMEZONE** is optional. It can be any valid time zone ID, such as PST or UTC. If it is not specified, the Oracle BI Publisher report time zone is applied implicitly.

To use the format-date function, the date from the data source must be in canonical format, which is YYYY-MM-DDThh:mm:ss+HH:MM where:

- YYYY is the year
- MM is the month
- DD is the day
- T is the separator between the date and time component
• hh is the hour in 24-hour format
• mm is the minutes
• ss is the seconds
• +HH:MM is the time zone offset from Universal Time (UTC), or Greenwich Mean Time

For example: 2011-05-26T09:30:10-07:00.

The data after the T is optional, therefore 2011-05-26 is a valid date.

Note:
If the time component and time zone offset are not included in the XML source date, Oracle BI Publisher assumes it represents 12:00 AM UTC (that is, YYYY-MM-DDT00:00:00-00:00).

In the Oracle BI Publisher and Oracle Fusion application integration environment, XDODEFDATE is used to pass the Oracle Fusion application date format pattern to Oracle BI Publisher, and XDODEFTIME can pass the Oracle Fusion application time format pattern to Oracle BI Publisher reports.

Consider that "yyyy-MM-dd (2011-05-27)" is selected for the Date Format, and "a hh:mm:ss (AM 12:49:13)" is selected for the Time Format in Oracle Fusion application Preferences, as shown in Figure 21-25.

Figure 21-25    Selecting Date and Time Format in Oracle Fusion Application Preferences

At runtime, XDODEFDATE is replaced with "yyyy-MM-dd" and XDODEFTIME is replaced with "a hh:mm:ss".

Therefore, to format date or timestamp values in Oracle BI Publisher reports with the Date Format or Time Format selected in Oracle Fusion application Preferences, the format-date function can be used:

• To display date only values
  
  ```xml
  <?format-date:Date_Only_Field; 'XDODEFDATE'?>
  ```

  Only the date is displayed, such as 2011-05-27.

• To display time only values
  
  ```xml
  <?format-date:Time_Only_Field; 'XDODEFTIME'?>
  ```

  Only the time is displayed, such as AM 12:49:13.

• To display datetime values
Both date and time are displayed, such as 2011-05-27 AM 12:49:13.

- To display System Date (sysdate)

```xml
<?format-date:xdoxslt:sysdate_as_xsdformat();' XDODEFDATE XDODEFTIME '?>
```


With these codes, the format of date or timestamp values varies according to the Date Format or the Time Format chosen in Oracle Fusion application Preferences.

Follow these steps to format date or timestamp values in the RTF template.

- Open the RTF template file in Microsoft Word.
- Choose **BI Publisher** > **Sample XML**, and choose a sample XML file that contains sample data for the number fields to be formatted in RTF template.
- Double-click the date or timestamp field to be modified, such as `ORDER_DATE`. In the BI Publisher Properties dialog, click the Properties tab and make sure that Type is set to Regular Text.
- Click the Advanced tab and enter format code, such as `<?format-date:ORDER_DATE; 'XDODEFDATE'??>`, as shown in Figure 21-26.

**Figure 21-26  Entering Date Format Code in Oracle BI Publisher Properties**

- Click **OK** and save the file.

Follow these steps to format date or timestamp values in graphs in Oracle BI Publisher reports.

- To set format masks for graphs, an Oracle Fusion application Oracle BI Publisher report designer needs to change the RTF template graph code manually.
- Edit the graph and change the graph definition code in the Advanced tab of the Graph dialog in template builder.
- Use an XSLT format such as `<xsl:value-of select="xdoxslt:xdo_format_date($_XDOXSLTCTX, current-`
group() / SGT_DATE, 'XDODEFDATE_XDODEFTIME')" /> to format a date-time value on the graph.

If you do not wish to format date or timestamps according to the Oracle Fusion application Date or Time format, you can use Oracle abstract format masks, such as:

- <?format-date:ORDER_DATE; 'SHORT'>
- <?format-date:ORDER_DATE; 'LONG'>
- <?format-date:xdoxslt:sysoxdate_as_xsdformat(); 'MEDIUM'>

With using abstract format masks, date and timestamp values are formatted according to the default format of the Oracle Fusion application user's locale.

21.10.4 How to Honor Time Zones in Oracle BI Publisher

In Oracle BI Publisher and Oracle Fusion application integration environment, the Time Zone selected in Oracle Fusion application Preferences is passed to the Oracle BI Publisher server and overrides the Oracle BI Publisher report time zone automatically. Therefore, if the TIMEZONE parameter is not specified in the format-date function, a date/time field in the Oracle BI Publisher report is formatted with the time zone selected in Oracle Fusion application Preferences. That is, time zone conversion happens between the Oracle BI Publisher server time zone and the Oracle Fusion application Preferences time zone.

If you do not wish to format a date-time field with the Oracle Fusion application Preferences Time Zone, you must explicitly specify a time zone ID in the format-date function:

<?format-date:Datetime_Field; 'XDODEFDATE_XDODEFTIME', 'timezone'>

where timezone is a static value, such as PST, UTC, or IST.

**Note:**

There is no parameter that passes the Oracle Fusion application Corporation Time Zone (CRTZ) or Legal Entity Time Zone (LETZ) to the Oracle BI Publisher server. Therefore, you cannot format a date/time value with CRTZ or LETZ dynamically by using a parameter to pass CRTZ or LETZ to Oracle BI Publisher reports. Instead, you have to hard-code the time zone ID within the format-date function.

21.11 Implement Localization Formatting in ADF Data Visualization Components

All numerical-based values need to be formatted correctly when they are presented to a user in Oracle Fusion applications graphs. For instance, a value of 1234.5 in the database might need to be displayed as 1 234,500 or 1,234.500 or 1.234,5. The user can change the number of digits found after the decimal digit, grouping separator, and decimal separator.

21.11.1 How to Format Numbers on a Graph

Before you begin, create an entity object and a view object with number fields, including any of the following attribute types: java.math.BigDecimal,
java.lang.Integer, or java.lang.Long. Drag and drop the entity and create a data visualization component such as a pie graph, bar graph or a line graph.

Numerical values on all parts of the graph need to be formatted according to the user’s number formatting preferences in Oracle Fusion applications. Developers need to add the tag `<af:convertNumber>` to all the tags in the graph where numerical values are displayed.

The ADF Data Visualization area graph supports number formatting on its Y1 and Y2 axis tick labels, and for marker text that appears on the data points. Users can customize the number formatting by adding `<af:convertNumber>` to the `<dvt:y1TickLabel>, <dvt:y2TickLabel>, <dvt:y1Format>, and <dvt:y2Format>` tags. Example 21-69 shows how to configure these tags.

In Figure 21-27, numbers such as 20,000 and 40,000 on the Y axis and numbers such as 12,003.67 on the tooltip are examples of numbers on an area graph that should be formatted according to the user’s number preferences in graphs in Oracle Fusion applications.

![Figure 21-27 Showing Number Formatting on an Area Graph](image)

Note:
For integer fields, use `pattern="#{applCorePrefs.integerFormatPattern}"` just like integer formatting in ADF Faces.

Generally, an attribute on an ordinal axis (O1 axis and X1 axis) of a graph in ADF Data Visualization is considered to be text, irrespective of its data type in the model or view layer. A developer can add `af:convertNumber` to such a view attribute by first adding `dvt:attributeFormat` to it, and only then will the resultant values on the x-axis and o-axis be formatted according to the user's number preferences. The `name="size"` entry refers to the categorical attribute (attribute on the ordinal/x axis). To format the integers, use `pattern="#{applCorePrefs.integerFormatPattern}"`, as shown in Example 21-70.

In Figure 21-28, numbers such as 2234 and 1014 are examples of numbers on the O axis of an area graph that should be formatted according to the user’s number preferences.
Figure 21-28  Numbers to Format Using User’s Preferences

Other Types of Graphs

Example 21-70 just gives an example of number formatting in area graphs. Table 21-1 presents the tags in which af:convertNumber can be added for number formatting in other types of graphs, such as bar graphs and pie charts, and the graphs in which dvt:attributeFormat can be used to format text attributes on the ordinal axis.

<table>
<thead>
<tr>
<th>Graph Type</th>
<th>Parent Tags for af:convertNumber</th>
<th>Is dvt:attributeFormat Needed for Number Formatting?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar Graph</td>
<td><a href="">dvt:y1TickLabel</a>, <a href="">dvt:y2TickLabel</a>, <a href="">dvt:y1Format</a>, and <a href="">dvt:y2Format</a></td>
<td>Yes (for O1 axis)</td>
</tr>
<tr>
<td>Bar Horizontal</td>
<td><a href="">dvt:y1TickLabel</a>, <a href="">dvt:y2TickLabel</a>, <a href="">dvt:y1Format</a>, and <a href="">dvt:y2Format</a></td>
<td>Yes (for O1 axis)</td>
</tr>
<tr>
<td>Bubble Graph</td>
<td><a href="">dvt:x1TickLabel</a>, <a href="">dvt:y1TickLabel</a>, <a href="">dvt:y2TickLabel</a>, <a href="">dvt:y1Format</a>, <a href="">dvt:y2Format</a> and <a href="">dvt:zFormat</a></td>
<td>Yes (but not for O1 axis, for X1 axis)</td>
</tr>
<tr>
<td>Combination graph</td>
<td><a href="">dvt:y1TickLabel</a>, <a href="">dvt:y2TickLabel</a>, <a href="">dvt:y1Format</a>, and <a href="">dvt:y2Format</a></td>
<td>Yes (for O1 axis)</td>
</tr>
<tr>
<td>Funnel Graph</td>
<td><a href="">dvt:sliceLabel</a></td>
<td>Yes (for funnel section values)</td>
</tr>
<tr>
<td>Line Graph</td>
<td><a href="">dvt:y1TickLabel</a>, <a href="">dvt:y2TickLabel</a>, <a href="">dvt:y1Format</a>, and <a href="">dvt:y2Format</a></td>
<td>Yes (for O1 axis)</td>
</tr>
<tr>
<td>Pareto Graph</td>
<td><a href="">dvt:y1TickLabel</a>, <a href="">dvt:y2TickLabel</a>, and <a href="">dvt:y1Format</a></td>
<td>Yes (for O1 axis)</td>
</tr>
<tr>
<td>Pie Graph</td>
<td><a href="">dvt:sliceLabel</a></td>
<td>Yes (for slice values)</td>
</tr>
<tr>
<td>Pie Bar Charts</td>
<td><a href="">dvt:y1TickLabel</a>, <a href="">dvt:sliceLabel</a> and <a href="">dvt:y1Format</a></td>
<td>Yes (for slice values)</td>
</tr>
<tr>
<td>Gantt Chart</td>
<td><a href="">af:column</a></td>
<td>Yes (for O1 axis)</td>
</tr>
<tr>
<td>Gauge</td>
<td><a href="">dvt:metricLabel</a> and <a href="">dvt:tickLabel</a></td>
<td>Yes (for O1 axis)</td>
</tr>
<tr>
<td>ADF Pivot Table</td>
<td><a href="">af:inputText</a> or <a href="">af:outputText</a> whose parent tag is <a href="">dvt:dataCell</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="">af:inputText</a> or <a href="">af:outputText</a> whose parent tag is <a href="">dvt:headerCell</a></td>
<td></td>
</tr>
</tbody>
</table>
21.11.2 Standards and Guidelines for Formatting Numbers in Graphs

- `dvt:numberFormat` has been deprecated and all Oracle Fusion applications developers should use only `af:convertNumber` to format numbers on graphs.

- Before adding an `af:convertNumber` tag to a graph, make sure that the `autoPrecision` attribute of the parent tag (for example `dvt:MarkerText`) is set to `off`. If `autoPrecision` is set to `on`, ADF Data Visualization does not follow the converter correctly, and formats numbers automatically, which will be incorrect in Oracle Fusion applications, as the number will not honor user preferences.

- Set `scaling="none"` before adding the `af:convertNumber` tag to a graph. Otherwise ADF Data Visualization will not follow the converter correctly.

21.11.3 How to Format Currency Values in ADF Data Visualization

Currency fields are represented by `java.lang.BigDecimal` in entity objects and view objects.

A currency field should always be formatted according to the currency code chosen in the context UI of the transaction.

For example, if the user selects "JPY" as the currency code from the context UI, the currency value should be formatted according to the Japanese Yen standard, whereas if USD is chosen as the currency code, the currency value should be formatted according to the American Dollar standard.

Oracle Applications Technology (ATG) provides two implementations to format numerical values according to the corresponding currency code:

- `fnd:currencyPattern(bindingToAmountCurrencyCode)`: This function formats the numerical value according to the currency code input as a parameter (for example, if "USD" is the input parameter, the value is formatted as "#,##0.00")

- `fnd:currencyPatternWithPrecisionAndSymbol(bindingToAmountCurrencyCode, bindingToAttrNamePrecision, bindingToAttrNameCurrencySymbol)`: This function formats the numerical value according to the currency code input as the first parameter; the second parameter indicates the number of decimal digits in the formatted value; the third parameter determines whether the formatted string shows the symbol for the currency code, the currency code itself or nothing after the numerical value. (The third parameter can be symbol, code or none.)
21.11.3.1 Formatting Currency Values on a Graph

Before you begin, create an entity object and a view object with number fields, including any of the following attribute types: java.math.BigDecimal, which is a currency value. Drag and drop the entity and create an ADF Data Visualization component, such as a pie graph, bar graph, or a line graph.

Numbers on the Y and O axis can be formatted as currency and can be displayed with the currency symbol, as shown in Example 21-71.

In Figure 21-29, number values such as 40,000.00 and 80,000.00 on the Y axis, and 13,344.22 on the tooltip, are examples of number fields in an area graph that should be formatted according to the currency code for these values. Numbers on the O1 axis should also be formatted according to the currency code. (In this case, for USD, the format is #,##0.00.)

Figure 21-29  Numbers to Format According to Currency Code

Other Types of Graphs

Example 21-71 gives an example of currency formatting in area graphs. Table 21-2 presents the tags in which af:convertNumber can be added for currency formatting in other types of graphs, and the graphs in which dvt:attributeFormat can be used to format text attributes on the ordinal axis.

Table 21-2  Using af:convertNumber for Currency Formatting and dvt:attributeFormat for Text Formatting

<table>
<thead>
<tr>
<th>Graph Type</th>
<th>Tags in which af:convertNumber Can Be Added for Currency Formatting</th>
<th>Can dvt:attributeFormat Be Added for Currency Formatting of Attributes on the Ordinal Axis?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar Graph</td>
<td><a href="">dvt:y1TickLabel</a>, <a href="">dvt:y2TickLabel</a>, <a href="">dvt:y1Format</a>, and <a href="">dvt:y2Format</a></td>
<td>Yes</td>
</tr>
<tr>
<td>Bar Horizontal</td>
<td><a href="">dvt:y1TickLabel</a>, <a href="">dvt:y2TickLabel</a>, <a href="">dvt:y1Format</a>, and <a href="">dvt:y2Format</a></td>
<td>Yes</td>
</tr>
<tr>
<td>Bubble Graph</td>
<td><a href="">dvt:x1TickLabel</a>, <a href="">dvt:y1TickLabel</a>, <a href="">dvt:y2TickLabel</a>, <a href="">dvt:x1Format</a>, <a href="">dvt:y1Format</a>, <a href="">dvt:y2Format</a> and <a href="">dvt:zFormat</a></td>
<td>Yes (but not for O1 axis, for X1 axis)</td>
</tr>
<tr>
<td>Combination Graph</td>
<td><a href="">dvt:y1TickLabel</a>, <a href="">dvt:y2TickLabel</a>, <a href="">dvt:y1Format</a>, and <a href="">dvt:y2Format</a></td>
<td>Yes</td>
</tr>
<tr>
<td>Funnel Graph</td>
<td><a href="">dvt:sliceLabel</a></td>
<td>Yes (for funnel section values)</td>
</tr>
<tr>
<td>Line Graph</td>
<td><a href="">dvt:y1TickLabel</a>, <a href="">dvt:y2TickLabel</a>, <a href="">dvt:y1Format</a>, and <a href="">dvt:y2Format</a></td>
<td>Yes</td>
</tr>
</tbody>
</table>
Table 21-2 (Cont.) Using af:convertNumber for Currency Formatting and dvt:attributeFormat for Text Formatting

<table>
<thead>
<tr>
<th>Graph Type</th>
<th>Tags in which af:convertNumber Can Be Added for Currency Formatting</th>
<th>Can dvt:attributeFormat Be Added for Currency Formatting of Attributes on the Ordinal Axis?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pareto Graph</td>
<td><a href="">dvt:y1TickLabel</a>, <a href="">dvt:y2TickLabel</a>, and <a href="">dvt:y1Format</a></td>
<td>Yes</td>
</tr>
<tr>
<td>Pie Graph</td>
<td><a href="">dvt:sliceLabel</a></td>
<td>Yes (for slice values)</td>
</tr>
<tr>
<td>Pie Bar Charts</td>
<td><a href="">dvt:y1TickLabel</a>, <a href="">dvt:sliceLabel</a> and <a href="">dvt:y1Format</a></td>
<td>Yes (for slice values)</td>
</tr>
<tr>
<td>Gantt Chart</td>
<td><a href="">af:column</a></td>
<td>No</td>
</tr>
<tr>
<td>Gauge</td>
<td><a href="">dvt:metricLabel</a> and <a href="">dvt:tickLabel</a></td>
<td>No</td>
</tr>
<tr>
<td>ADF Pivot Table</td>
<td><a href="">af:inputText</a>/<a href="">af:outputText</a> whose parent tag is <a href="">dvt:dataCell</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="">af:inputText</a>/<a href="">af:outputText</a> whose parent tag is <a href="">dvt:headerCell</a></td>
<td></td>
</tr>
</tbody>
</table>

Example 21-71  Area Graph Y1 Axis Currency Value Formatting

```xml
<dvt:y1TickLabel>
  <af:convertNumber pattern="#{ fnd:currencyPattern(YourCurrencyCode)}"/>
</dvt:y1TickLabel>
<dvt:markerText><dvt:y1Format>
  <af:convertNumber pattern="#{ fnd:currencyPattern(YourCurrencyCode)}"/>
</dvt:y1Format></dvt:markerText>
```

21.11.3.2 Standards and Guidelines for Formatting Currency Values in Graphs

- Before adding an af:convertNumber tag to a graph, make sure that the autoPrecision attribute of the parent tag (such as dvt:MarkerText) is set to off. If autoPrecision is set to on, ADF Data Visualization does not follow the converter correctly, and instead formats numbers automatically, which will be incorrect in Oracle Fusion applications, as the number will not honor user preferences.
- Set scaling="none" before adding the af:convertNumber tag to a graph. Otherwise, ADF Data Visualization will not follow the converter correctly.

21.11.4 How to Format Dates and Timestamp Values in ADF Data Visualization

All date values on a graph need to be formatted correctly when they are presented to a user. Users expect that they can input values according to their date formatting preferences. For instance, a value of 13th August, 2011 in the database might have to be displayed as 08/13/2011 for a certain user. A similar example of a date-time value is that a value of 26th July 2011, 2:00:00 PM might have to be displayed as 14:00 07/26/2011.

21.11.4.1 Formatting Dates and Timestamp Values on a Graph

Before you begin, create an entity object and a view object with date or timestamp fields, including any of these attribute types: java.sql.Date or java.sql.Timestamp. Drag
and drop the entity and create an ADF Data Visualization component, such as a pie graph, a bar graph or a line graph.

A graph typically uses categorical attributes, such as product, geography or year, on an ordinal axis (x axis or dvt:o1Axis) and for the marker tooltips. For the dvt:o1 axis, the data is considered as text no matter what its type in the underlying layers. Therefore, any number or date on the dvt:o1 axis does not honor any formatting.

A user can add af:convertNumber/af:convertDateTime by first adding dvt:attributeFormat to it, and the resultant values on the x-axis will be formatted, as shown in Example 21-72. Also, with this code, wherever the ordinal axis values are used in the user interface, they will appear with the proper formatting.

---

**Note:**

- If there is a single categorical Date attribute being displayed on the O1Axis, the graph displays a TimeAxis instead of the typical O1 / Ordinal Axis. The TimeAxis will show dates in a hierarchical format instead of as a single label on an O1 Axis, for example June 27, 2001. To show a single label on the O1 Axis, the TimeAxis should be turned off (timeAxisType="TAT_OFF") and a <dvt:attributeFormat> is necessary to specify the date format.

- The Area graph displays a time axis when dates (object type java.util.Date) are specified for the column labels. Several timeXXX attributes are defined on the graph tag to customize the time axis. The child tag timeAxisDateFormat controls the format in which the time axis labels are displayed.

  <dvt:timeAxisDateFormat dayFormat="DAY_OF_MONTH" monthFormat="MONTH_LONG" quarterFormat="NONE" yearFormat="YEAR_LONG" timeFormat="HOUR_MINUTE_SECOND"/>

---

Example 21-73 shows timestamp formatting on the O1 axis.

This is the same as the case of ADF Faces, in which the UPTZPattern is used to format timestamp values and dateFormatPattern is used to format date values.

In Figure 21-30, dates such as 2/1/2015 on the O1 axis and 8/9/2009 on the tooltip are examples of date fields on an area graph that should be formatted according to the user’s date formatting preferences.

**Figure 21-30   Dates to Format Using User Preferences**

![Image showing area graph with dates formatted as per user preferences.](image)

Other Types of Graphs

Example 21-73 just gives an example of date and date-time formatting in area graphs. **Table 21-3** shows the tags in which af:convertDateTime can be added for date and timestamp formatting in the other types of graphs.
### Table 21-3  Using `af:convertDateTime` for Date and Timestamp Formatting

<table>
<thead>
<tr>
<th>Graph Type</th>
<th>Is <code>dvt:attributeFormat</code> Required?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar Graph</td>
<td>Yes</td>
</tr>
<tr>
<td>Bar Horizontal</td>
<td>Yes</td>
</tr>
<tr>
<td>Bubble Graph</td>
<td>Yes</td>
</tr>
<tr>
<td>Combination Graph</td>
<td>Yes</td>
</tr>
<tr>
<td>Funnel Graph</td>
<td>Yes (for funnel section values)</td>
</tr>
<tr>
<td>Line Graph</td>
<td>Yes</td>
</tr>
<tr>
<td>Pareto Graph</td>
<td>Yes</td>
</tr>
<tr>
<td>Pie Graph</td>
<td>Yes</td>
</tr>
<tr>
<td>Pie Bar Charts</td>
<td>Yes</td>
</tr>
<tr>
<td>Gantt Chart</td>
<td>No (Add <code>af:convertDateTime</code> as a child tag of <code>af:column</code>)</td>
</tr>
<tr>
<td>Gauge</td>
<td>N/A</td>
</tr>
<tr>
<td>ADF Pivot Table</td>
<td><code>&lt;af:inputText&gt;/&lt;af:outputText&gt;</code> whose parent tag is <code>&lt;dvt:dataCell&gt;</code>/<a href="">af:inputText</a>/<a href="">af:outputText</a><code>whose parent tag is</code><a href="">dvt:headerCell</a>`</td>
</tr>
</tbody>
</table>

**Example 21-72  Date Formatting on the O1 Axis**

```xml
<dvt:attributeFormat id="af1" name="OrderDate">
  <af:convertDateTime type="date" pattern="#{applCorePrefs.dateFormatPattern}"/>
</dvt:attributeFormat>
```

**Example 21-73  Timestamp Formatting on the O1 Axis**

```xml
<dvt:attributeFormat id="af1" name="LastUpdateDate">
  <af:convertDateTime type="both" pattern="#{applCorePrefs.UPTZPattern}"/>
</dvt:attributeFormat>
```

## 21.12 Configure National Language Support Attributes

In Oracle Fusion Applications, National Language Support (NLS) refers to the ability to run an application instance in any single supported language, including specific regional or territorial number and date formats. Typically, to support a given language, only the customer-facing components of the software (user interface, lookup tables, online documentation, and so on) are translated. Translations are delivered via NLS patches.

### 21.12.1 Session National Language Support Attributes

Oracle Fusion Applications manage NLS attributes at the session level. At runtime, these attributes are initialized based on the user’s profile, and are applied when needed by Applications Core. For example, the session date format mask is initialized based on the user’s preferred date format mask. The date format mask is automatically applied when date is rendered and parsed. As such, it is unnecessary to manually specify NLS attributes in design time.
In certain situations, however, you may need to access the NLS attributes for the purposes of data formatting or parsing your code. To do so, use the managed bean `ApplCorePrefs`.

Table 21-4 lists the Oracle Fusion Applications session NLS attributes, the profile used to set each attribute and the possible values for session attributes.

<table>
<thead>
<tr>
<th>Session Attribute</th>
<th>Profile</th>
<th>Values</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>LANGUAGE</td>
<td>FND_LANGUAGE</td>
<td>select DESCRIPTION, LANGUAGE_TAG from FND_LANGUAGES_B where INSTALLED_FLAG in ('I', 'B');</td>
<td>Primary attribute used to represent the current language. Corresponds to the LANGUAGE_TAG column in FND_LANGUAGES. Looks up the corresponding NLS_LANGUAGE and alters the session in the database. Valid examples are es, es-US, fr.</td>
</tr>
<tr>
<td>NLS_LANG</td>
<td></td>
<td>Represents the two letter language code, which is derived using the LANGUAGE attributes. This value is derived rather than explicitly set.</td>
<td></td>
</tr>
<tr>
<td>NLS_LANGUAGE</td>
<td></td>
<td>Represents the NLS language, which is derived from the LANGUAGE attribute. This value is derived rather than explicitly set.</td>
<td></td>
</tr>
<tr>
<td>NLS_SORT</td>
<td>FND_NLS_SORT</td>
<td>select MEANING, LOOKUP_CODE from FND_LOOKUPS where LOOKUP_TYPE = 'NLS_SORT' and ENABLED_FLAG = 'Y' and SYSDATE between START_DATE_ACTIVE and nvl(END_DATE_ACTIVE, SYSDATE +1);</td>
<td>Setting this attribute results in an altered session in the database for the NLS_SORT database attribute.</td>
</tr>
<tr>
<td>DATE_FORMAT</td>
<td>FND_DATE_FORMAT</td>
<td>select MEANING, LOOKUP_CODE from FND_LOOKUPS where LOOKUP_TYPE = 'DATE_FORMAT' and ENABLED_FLAG = 'Y' and SYSDATE between START_DATE_ACTIVE and nvl(END_DATE_ACTIVE, SYSDATE +1);</td>
<td></td>
</tr>
<tr>
<td>Session Attribute</td>
<td>Profile</td>
<td>Values</td>
<td>Comments</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------</td>
<td>------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TIME_FORMAT</td>
<td>FND_TIME_FORMAT</td>
<td>select MEANING, LOOKUP_CODE from FND_LOOKUPS where LOOKUP_TYPE = 'TIME_FORMAT' and ENABLED_FLAG = 'Y' and SYSDATE between START_DATE_ACTIVE and nvl(END_DATE_ACTIVE, SYSDATE +1);</td>
<td></td>
</tr>
<tr>
<td>GROUPING_SEPARATOR</td>
<td>FND_GROUPING_SEPARATOR</td>
<td>select MEANING, LOOKUP_CODE from FND_LOOKUPS where LOOKUP_TYPE = 'GROUPING_SEPARATOR' and ENABLED_FLAG = 'Y' and SYSDATE between START_DATE_ACTIVE and nvl(END_DATE_ACTIVE, SYSDATE +1);</td>
<td></td>
</tr>
<tr>
<td>DECIMAL_SEPARATOR</td>
<td>FND_DECIMAL_SEPARATE</td>
<td>select MEANING, LOOKUP_CODE from FND_LOOKUPS where LOOKUP_TYPE = 'DECIMAL_SEPARATOR' and ENABLED_FLAG = 'Y' and SYSDATE between START_DATE_ACTIVE and nvl(END_DATE_ACTIVE, SYSDATE +1);</td>
<td></td>
</tr>
<tr>
<td>CURRENCY</td>
<td>FND_CURRENCY</td>
<td>select NAME, CURRENCY_CODE from FND_CURRENCIES_VL where ENABLED_FLAG = 'Y' and SYSDATE between START_DATE_ACTIVE and nvl(END_DATE_ACTIVE, SYSDATE +1);</td>
<td>This attribute specifies the preferred currency code. It has no corresponding database attribute.</td>
</tr>
<tr>
<td>TERRITORY</td>
<td>FND_TERRITORY</td>
<td>select TERRITORY_SHORT_NAME, TERRITORY_CODE from FND_TERRITORIES_VL where ENABLED_FLAG = 'Y';</td>
<td>This attribute specifies the preferred territory. This attribute differs from the database attribute NLS_TERRITORY, as Oracle Fusion Applications supports more territories than the database. The database attribute is permanently set to the value AMERICAN.</td>
</tr>
</tbody>
</table>
Table 21-4 (Cont.) Session NLS Attributes

<table>
<thead>
<tr>
<th>Session Attribute</th>
<th>Profile</th>
<th>Values</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMEZONE</td>
<td>FND_TIMEZONE</td>
<td>select TIMEZONE_CODE, NAME from FND_TIMEZONES_VL where ENABLED_FLAG = 'Y';</td>
<td>This attribute specifies the preferred time zone value.</td>
</tr>
<tr>
<td>CLIENT_ENCODING</td>
<td>FND_CLIENT_ENCODING</td>
<td>select MEANING, LOOKUP_CODE from FND_LOOKUPS where LOOKUP_TYPE = 'CLIENT_ENCODING' and ENABLED_FLAG = 'Y' and SYSDATE between START_DATE_ACTIVE and nvl(END_DATE_ACTIVE, SYSDATE +1);</td>
<td></td>
</tr>
</tbody>
</table>

Table 21-5 lists language and territory values used with NLS attributes.

Table 21-5 Language and Territory Values

<table>
<thead>
<tr>
<th>LANGUAGE_TAG_CODE</th>
<th>LANGUAGE</th>
<th>LANGUAGE_ID</th>
<th>NLS_LANGUAGE_ID</th>
<th>NLS_TERRITORY</th>
<th>ISO_LANGUAGE</th>
<th>ISO_TERRITORY</th>
<th>NLS_CODESET</th>
<th>ISO_LANGUAGE_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ar</td>
<td>AR</td>
<td>8</td>
<td>ARABIC</td>
<td>UNITED ARAB EMIRATES</td>
<td>ar</td>
<td>AE</td>
<td>AR8ISO8859P6</td>
<td>ara</td>
</tr>
<tr>
<td>bg</td>
<td>BG</td>
<td>101</td>
<td>BULGARIA</td>
<td>BULGARIA</td>
<td>bg</td>
<td>BG</td>
<td>CL8ISO8859P5</td>
<td>bul</td>
</tr>
<tr>
<td>cs</td>
<td>CS</td>
<td>30</td>
<td>CZECH</td>
<td>CZECH REPUBLIC</td>
<td>cs</td>
<td>CZ</td>
<td>EE8ISO8859P2</td>
<td>ces</td>
</tr>
<tr>
<td>de</td>
<td>D</td>
<td>4</td>
<td>GERMAN</td>
<td>GERMANY</td>
<td>de</td>
<td>DE</td>
<td>WE8ISO8859P1</td>
<td>deu</td>
</tr>
<tr>
<td>da</td>
<td>DK</td>
<td>5</td>
<td>DANISH</td>
<td>DENMARK</td>
<td>da</td>
<td>DK</td>
<td>WE8ISO8859P1</td>
<td>dan</td>
</tr>
<tr>
<td>es</td>
<td>E</td>
<td>11</td>
<td>SPANISH</td>
<td>SPAIN</td>
<td>es</td>
<td>ES</td>
<td>WE8ISO8859P1</td>
<td>spa</td>
</tr>
<tr>
<td>eg</td>
<td>EG</td>
<td>118</td>
<td>EGYPTIAN</td>
<td>EGYPT</td>
<td>eg</td>
<td>EG</td>
<td>AR8ISO8859P6</td>
<td>egy</td>
</tr>
<tr>
<td>el</td>
<td>EL</td>
<td>104</td>
<td>GREEK</td>
<td>GREECE</td>
<td>el</td>
<td>GR</td>
<td>EL8ISO8859P7</td>
<td>ell</td>
</tr>
<tr>
<td>es-US</td>
<td>ESA</td>
<td>29</td>
<td>LATIN AMERICAN</td>
<td>AMERICA</td>
<td>es</td>
<td>US</td>
<td>WE8ISO8859P1</td>
<td>spa</td>
</tr>
<tr>
<td>fr</td>
<td>F</td>
<td>2</td>
<td>FRENCH</td>
<td>FRANCE</td>
<td>fr</td>
<td>FR</td>
<td>WE8ISO8859P1</td>
<td>fra</td>
</tr>
<tr>
<td>fr-CA</td>
<td>FRC</td>
<td>3</td>
<td>CANADIAN FRENCH</td>
<td>CANADA</td>
<td>fr</td>
<td>CA</td>
<td>WE8ISO8859P1</td>
<td>fra</td>
</tr>
<tr>
<td>en-GB</td>
<td>GB</td>
<td>1</td>
<td>ENGLISH</td>
<td>UNITED KINGDOM</td>
<td>en</td>
<td>GB</td>
<td>WE8ISO8859P1</td>
<td>eng</td>
</tr>
<tr>
<td>LANGUAGE_TAG</td>
<td>LANGUAGE</td>
<td>LANGUAGE_ID</td>
<td>NLS_LANGUGE_ID</td>
<td>NLS_TERRITORY</td>
<td>ISO_LANGUAGE</td>
<td>ISO_TERRITORY</td>
<td>NLS_CODESET</td>
<td>ISO_LANGUAGE_3</td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
<td>-------------</td>
<td>----------------</td>
<td>---------------</td>
<td>--------------</td>
<td>---------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>hr</td>
<td>HR</td>
<td>103</td>
<td>CROATIAN</td>
<td>CROATIA</td>
<td>hr</td>
<td>HR</td>
<td>EE8ISO8859P2</td>
<td>hrv</td>
</tr>
<tr>
<td>hu</td>
<td>HU</td>
<td>28</td>
<td>HUNGARIAN</td>
<td>HUNGARY</td>
<td>hu</td>
<td>HU</td>
<td>EE8ISO8859P2</td>
<td>hun</td>
</tr>
<tr>
<td>it</td>
<td>I</td>
<td>108</td>
<td>ITALIAN</td>
<td>ITALY</td>
<td>it</td>
<td>IT</td>
<td>WE8ISO8859P1</td>
<td>ita</td>
</tr>
<tr>
<td>is</td>
<td>IS</td>
<td>106</td>
<td>ICELANDIAN</td>
<td>ICELAND</td>
<td>is</td>
<td>IS</td>
<td>WE8ISO8859P1</td>
<td>isl</td>
</tr>
<tr>
<td>he</td>
<td>IW</td>
<td>107</td>
<td>HEBREW</td>
<td>ISRAEL</td>
<td>he</td>
<td>IL</td>
<td>IW8ISO8859P8</td>
<td>heb</td>
</tr>
<tr>
<td>ja</td>
<td>JA</td>
<td>15</td>
<td>JAPANESE</td>
<td>JAPAN</td>
<td>ja</td>
<td>JP</td>
<td>JA16EUC</td>
<td>jpn</td>
</tr>
<tr>
<td>ko</td>
<td>KO</td>
<td>16</td>
<td>KOREAN</td>
<td>KOREA</td>
<td>ko</td>
<td>KR</td>
<td>KO16KSC5601</td>
<td>kor</td>
</tr>
<tr>
<td>lt</td>
<td>LT</td>
<td>109</td>
<td>LITHUANIAN</td>
<td>LITHUANIA</td>
<td>lt</td>
<td>LT</td>
<td>NEE8ISO8859P4</td>
<td>lit</td>
</tr>
<tr>
<td>no</td>
<td>N</td>
<td>10</td>
<td>NORWEGIAN</td>
<td>NORWAY</td>
<td>no</td>
<td>NO</td>
<td>WE8ISO8859P1</td>
<td>nor</td>
</tr>
<tr>
<td>nl</td>
<td>NL</td>
<td>6</td>
<td>DUTCH</td>
<td>THE NETHERLANDS</td>
<td>nl</td>
<td>NL</td>
<td>WE8ISO8859P1</td>
<td>nld</td>
</tr>
<tr>
<td>pl</td>
<td>PL</td>
<td>110</td>
<td>POLISH</td>
<td>POLAND</td>
<td>pl</td>
<td>PL</td>
<td>EE8ISO8859P2</td>
<td>pol</td>
</tr>
<tr>
<td>pt</td>
<td>PT</td>
<td>18</td>
<td>PORTUGUESE</td>
<td>PORTUGAL</td>
<td>pt</td>
<td>PT</td>
<td>WE8ISO8859P1</td>
<td>por</td>
</tr>
<tr>
<td>pt-BR</td>
<td>PTB</td>
<td>26</td>
<td>BRAZILIAN</td>
<td>PORTUGUESE</td>
<td>t</td>
<td>BR</td>
<td>WE8ISO8859P1</td>
<td>por</td>
</tr>
<tr>
<td>ro</td>
<td>RO</td>
<td>111</td>
<td>ROMANIAN</td>
<td>ROMANIA</td>
<td>ro</td>
<td>RO</td>
<td>EE8ISO8859P2</td>
<td>ron</td>
</tr>
<tr>
<td>ru</td>
<td>RU</td>
<td>112</td>
<td>RUSSIAN</td>
<td>RUSSIA</td>
<td>ru</td>
<td>RU</td>
<td>CL8ISO8859P5</td>
<td>rus</td>
</tr>
<tr>
<td>sv</td>
<td>S</td>
<td>13</td>
<td>SWEDISH</td>
<td>SWEDEN</td>
<td>sv</td>
<td>SE</td>
<td>WE8ISO8859P1</td>
<td>swe</td>
</tr>
<tr>
<td>fi</td>
<td>SF</td>
<td>7</td>
<td>FINNISH</td>
<td>FINLAND</td>
<td>fi</td>
<td>FI</td>
<td>WE8ISO8859P1</td>
<td>fin</td>
</tr>
<tr>
<td>sk</td>
<td>SK</td>
<td>113</td>
<td>SLOVAK</td>
<td>SLOVAKIA</td>
<td>sk</td>
<td>SK</td>
<td>EE8ISO8859P2</td>
<td>slk</td>
</tr>
<tr>
<td>sl</td>
<td>SL</td>
<td>114</td>
<td>SLOVENIAN</td>
<td>SLOVENIA</td>
<td>sl</td>
<td>SI</td>
<td>EE8ISO8859P2</td>
<td>slv</td>
</tr>
<tr>
<td>th</td>
<td>TH</td>
<td>115</td>
<td>THAI</td>
<td>THAILAND</td>
<td>th</td>
<td>TH</td>
<td>TH8TISASCII</td>
<td>tha</td>
</tr>
<tr>
<td>tr</td>
<td>TR</td>
<td>116</td>
<td>TURKISH</td>
<td>TURKEY</td>
<td>tr</td>
<td>TR</td>
<td>WE8ISO8859P9</td>
<td>tur</td>
</tr>
<tr>
<td>en</td>
<td>US</td>
<td>0</td>
<td>AMERICAN</td>
<td>AMERICA</td>
<td>en</td>
<td>US</td>
<td>US7ASCII</td>
<td>eng</td>
</tr>
<tr>
<td>zh-CN</td>
<td>ZHS</td>
<td>14</td>
<td>SIMPLIFIED CHINESE</td>
<td>CHINA</td>
<td>zh</td>
<td>CN</td>
<td>ZHS16CGB231280</td>
<td>zho</td>
</tr>
<tr>
<td>zh-TW</td>
<td>ZHT</td>
<td>117</td>
<td>TRADITIONAL CHINESE</td>
<td>TAIWAN</td>
<td>zh</td>
<td>TW</td>
<td>ZHT16BIG5</td>
<td>zho</td>
</tr>
<tr>
<td>sq</td>
<td>SQ</td>
<td>67</td>
<td>ALBANIAN</td>
<td>ALBANIA</td>
<td>sq</td>
<td>AL</td>
<td>EE8ISO8859P2</td>
<td>sqi</td>
</tr>
</tbody>
</table>
### Table 21-5  (Cont.) Language and Territory Values

<table>
<thead>
<tr>
<th>LANGUAGE_CODE</th>
<th>LANGUAGE_ID</th>
<th>LANGUAGE</th>
<th>TERRITORY</th>
<th>ISO_LANGUAGE</th>
<th>ISO_TERRITORY</th>
<th>NLS_CODESET</th>
<th>ISO_LANGUAGE_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>vi</td>
<td>VN</td>
<td>43</td>
<td>VIETNAMESE</td>
<td>VIETNAM</td>
<td>vi</td>
<td>VN</td>
<td>VN8MSWIN1258</td>
</tr>
<tr>
<td>id</td>
<td>IN</td>
<td>46</td>
<td>INDONESIAN</td>
<td>INDONESIA</td>
<td>id</td>
<td>ID</td>
<td>WE8ISO8859P1</td>
</tr>
</tbody>
</table>

#### 21.12.2 Database Session Attributes

Oracle Fusion Applications does not use most of the database session NLS parameters. Instead, these are set to constant values such that typically, the user’s preferred values are not reflected. This is true for most parameters except the following: NLS_LANGUAGE which is set to view link view access, and NLS_SORT, which is set to use the database linguistic sorting functionality.

The parameters TO_NUMBER, TO_DATE, TO_TIMESTAMP and TO_CHAR are used to format and parse SQL or PL/SQL statements. These parameters are based on constant values, the canonical format. The parameter FND_DATE for PL/SQL packages also works with the canonical format. Formatting and parsing should be done at the presentation, rather than the model layer.

Oracle Fusion Applications session management controls database session parameters. As such, the database NLS session parameter values must not be altered.

Table 21-6 lists the following:

- **Database attribute**: The database NLS session parameter name.
- **Associated session attribute**: The Oracle Fusion Applications NLS session attribute name related to the database attribute name, if one exists. If the attributes are indeed related, configuring a value for the Oracle Fusion Applications NLS session attribute results in an ALTER SESSION in the database layer.
- **Default value**: The default value of the attribute. This value is configured both in the database and init.ora as the database default.
- **Alter Session**: Indicates whether an ALTER SESSION is created in the database. When an ALTER SESSION initiates at session creation or attachment, execute NLS_LANGUAGE and NLS_TERRITORY first, as these may affect other attributes.
  - **Create**: An ALTER SESSION is created once, when the connection is first created.
  - **Update**: The ALTER SESSION updates when the session attaches, or whenever an associated attribute value changes mid-session.
  - **Never**: An ALTER SESSION is not created, and the default database value is unchanged.
<table>
<thead>
<tr>
<th>Database Attribute</th>
<th>Default Value</th>
<th>Alter Session</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLS_CALENDAR</td>
<td>None</td>
<td>Never</td>
<td>This attribute need not be set as the default value is GREGORIAN. Accept the default value.</td>
</tr>
<tr>
<td>NLS_COMP</td>
<td>None</td>
<td>Never</td>
<td>This attribute need not be set as the default value is BINARY. Accept the default value.</td>
</tr>
<tr>
<td>NLS_CURRENCY</td>
<td>None</td>
<td>Never</td>
<td>Accept the default value.</td>
</tr>
<tr>
<td>NLS_DATE_FORMAT</td>
<td>YYYY-MM-DD</td>
<td>Create</td>
<td>Fixed value.</td>
</tr>
<tr>
<td>NLS_DATE_LANGUAGE</td>
<td>NUMERIC DATE LANGUAGE</td>
<td>Create</td>
<td>Fixed value. Does not require an attribute or profile.</td>
</tr>
<tr>
<td>NLS_ISO_CURRENCY</td>
<td>None</td>
<td>Never</td>
<td>Accept the default value.</td>
</tr>
<tr>
<td>NLS_LANGUAGE</td>
<td>None</td>
<td>Update</td>
<td>The value of this attribute is fixed. See the LANGUAGE attribute in Table 21-4.</td>
</tr>
<tr>
<td>NLS_TERRITORY</td>
<td>AMERICA</td>
<td>Create</td>
<td>The value of this attribute is fixed. See the TERRITORY attribute in Table 21-4.</td>
</tr>
<tr>
<td>NLS_LENGTH_SEMANTICS</td>
<td>CHAR</td>
<td>Create</td>
<td>Fixed value. Does not require an attribute or profile.</td>
</tr>
<tr>
<td>NLS_NCHAR_CONV_EXCP</td>
<td>None</td>
<td>Never</td>
<td>Accept the default value. This parameter is not used.</td>
</tr>
<tr>
<td>NLS_NUMERIC_CHARACTERS</td>
<td>, .</td>
<td>Create</td>
<td>Fixed value. Choose group and decimal separators independently.</td>
</tr>
<tr>
<td>NLS_SORT</td>
<td>None</td>
<td>Update</td>
<td>To enable linguistic sorting, the NLS_SORT session attribute is used to set the value for this database attribute. See the NLS_SORT session attribute in Table 21-4.</td>
</tr>
<tr>
<td>NLS_TIME_FORMAT</td>
<td>HH24:MI:SS.FF</td>
<td>Create</td>
<td>Fixed value. Does not require an attribute or profile.</td>
</tr>
<tr>
<td>NLS_TIME_TZ_FORMAT</td>
<td>HH24:MI:SS.FF TZR</td>
<td>Create</td>
<td>Fixed value. Does not require an attribute or profile.</td>
</tr>
<tr>
<td>NLS_TIMESTAMP_FORMAT</td>
<td>YYYY-MM-DD HH24:MI:SS.FF</td>
<td>Create</td>
<td>Fixed value. Does not require an attribute or profile.</td>
</tr>
<tr>
<td>NLS_TIMESTAMP_TZ_FORMAT</td>
<td>YYYY-MM-DD HH24:MI:SS.FF TZR</td>
<td>Create</td>
<td>Fixed value. Does not require an attribute or profile.</td>
</tr>
<tr>
<td>NLS_DUAL_CURRENCY</td>
<td>None</td>
<td>Never</td>
<td>Accept the default value.</td>
</tr>
</tbody>
</table>
As the language attributes tracked on the session reflect Java values, you cannot use them for formatting on the PL/SQL layer.

### 21.12.3 Session NLS Attributes

The NLS attributes are managed in the Oracle Middleware Extensions for Applications layer and initialized based on preferences defined in the profiles.

<table>
<thead>
<tr>
<th>Session Attribute</th>
<th>Corresponding Profile</th>
<th>applCorePrefs Attribute</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>LANGUAGE</td>
<td>FND_LANGUAGE</td>
<td>locale</td>
<td>This is the primary attribute that is used to represent the current language. RFC-4646 language tag. The locale attribute gives a Java locale corresponding to the session language.</td>
</tr>
<tr>
<td>LS_LANG</td>
<td></td>
<td></td>
<td>This value represents the two-letter Oracle language code, and is derived based on the LANGUAGE setting. As such, it cannot be set and eventually may no longer be listed explicitly as a separate session attribute.</td>
</tr>
<tr>
<td>NLS_LANGUAGE</td>
<td></td>
<td></td>
<td>This value represents Oracle NLS language, and is derived from the LANGUAGE setting. As such, it cannot be set and eventually may no longer be listed explicitly as a separate session attribute.</td>
</tr>
<tr>
<td>NLS_SORT</td>
<td>FND_NLS_SORT</td>
<td></td>
<td>Setting this attribute results in an altered session in the database for the NLS_SORT database attribute. The value is the sort name of the Oracle Database.</td>
</tr>
<tr>
<td>DATE_FORMAT</td>
<td>FND_DATE_FORMAT</td>
<td>dateFormatPattern</td>
<td>This is the preferred date format in Java date format string defined in SimpleDateFormat.</td>
</tr>
</tbody>
</table>
Table 21-7  (Cont.) NLS Attributes

<table>
<thead>
<tr>
<th>Session Attribute</th>
<th>Corresponding Profile</th>
<th>applCorePrefs Attribute</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME_FORMAT</td>
<td>FND_TIME_FORMAT</td>
<td>UPTZPattern</td>
<td>The preferred time format in Java time format string defined in SimpleDateFormat. UPTZPattern is a combination of DATE_FORMAT and TIME_FORMAT with time zone information.</td>
</tr>
<tr>
<td>GROUPING_SEPARATOR</td>
<td>ND_GROUPING_SEPARATOR</td>
<td>numberGroupingSeparator</td>
<td>This attribute is the session grouping separator.</td>
</tr>
<tr>
<td>DECIMAL_SEPARATOR</td>
<td>FND_DECIMAL_SEPARATOR</td>
<td>decimalSeparator</td>
<td>This attribute is the session decimal separator.</td>
</tr>
<tr>
<td>CURRENCY</td>
<td>FND_CURRENCY</td>
<td></td>
<td>The preferred currency code. No corresponding RDBMS attribute.</td>
</tr>
<tr>
<td>TERRITORY</td>
<td>FND_TERRITORY</td>
<td></td>
<td>The preferred territory in ISO 3166 alpha-2 code. Differs from the RDBMS NLS_TERRITORY attribute, as Oracle Fusion Applications supports more territories than the database. The database attribute is fixed at AMERICAN and never changes.</td>
</tr>
<tr>
<td>TIMEZONE</td>
<td>FND_TIMEZONE</td>
<td>UPTZ</td>
<td>The preferred time zone. No corresponding RDBMS attribute.</td>
</tr>
<tr>
<td>CLIENT_ENCODING</td>
<td>FND_CLIENT_ENCODING</td>
<td></td>
<td>The preferred encoding for file uploads, downloads and exports in Java encoding name.</td>
</tr>
</tbody>
</table>

21.13 Standards and Guidelines for Localization Formatting

The following standards and guidelines apply to localization formatting:

- After dragging and dropping a view object onto a JSF page, the Oracle ADF tags for that view object are set. If you change the attributes of the view object after you create the JSF page, new Oracle ADF tags are not created. You must make these changes manually by editing the tags.

- You can change tags generated by Applications Core at design time by editing them manually.
This chapter discusses the basics of using flexfields to enable customers to add custom attributes to a business object in an Oracle Fusion application.

This chapter includes the following sections:

- Introduction to Flexfields
- Participant Roles
- The Flexfield Development Lifecycle
- Flexfields in the Application User Interface

22.1 Introduction to Flexfields

In your role as a developer, it is often impossible for you to anticipate all the database columns and UI fields your customers might need, or how each field should look as end user needs change. Flexfields enable customers to configure their applications to meet their business needs without having to perform custom programming.

The basic premise of a flexfield is to encapsulate all of the pieces of information related to a specific purpose, such as the components of a student's contact information, or the features of a product in inventory, or a key identifying a particular purchase by a particular person for a particular company on a particular date. A flexfield is a set of placeholder fields, called segments, that are associated with a business object. A segment captures a single atomic value, which is represented in the application database as a single column. In the application UI, a flexfield's segments can be presented as individual table columns, as separate fields, or as a concatenated string of values.

Those performing in the role of an implementor, either on behalf of the developer organization for a developer flexfield or on behalf of a customer for a customer flexfield, configure a flexfield by specifying the prompt, length, and data type of each flexfield segment. Configuration includes the specification of valid values for each segment, and the meaning of each value. Configuration also involves defining structure and context.

There are three types of flexfields, all of which enable implementors to configure application features without programming. These configurations are fully supported within Oracle Fusion Applications:

- Descriptive: A descriptive flexfield gives customers the ability to extend a data model with additional attributes. A descriptive flexfield provides a fixed number of segments for a business object, which the customer can optionally use to add custom attributes to meet their business needs, define how the attributes are validated, and configure how the attributes are displayed.
• Extensible: An extensible flexfield is similar to a descriptive flexfield, but with the following additional features:
  – The number of configurable segments is not fixed. That is, the number of segments is extensible.
  – Attributes can be grouped into a context so they will always be presented together in the application user interface.
  – Hierarchical categories enable implementors to reuse contexts for similar entities.
  – Extensible flexfields support one-to-many relationships between the entity and the extended attribute rows.

• Key: A key flexfield enables customers to use their own coding scheme or naming scheme to identify their business entities by giving implementors the ability to configure the business entities to be identified using flexible, multipart, intelligent key codes. Each element (segment) of the key may be individually meaningful, as well as the combination as a whole. For example, key flexfields might represent part numbers and account numbers. Key flexfields are never optional; customers must configure them for their Oracle Fusion applications to operate correctly.

To better understand flexfields and the differences among descriptive flexfields, extensible flexfields, and key flexfields, it is important to understand flexfield structures and contexts.

• Structure: A flexfield structure is a specific configuration of segments. If you add or remove segments, or, in the case of key flexfields, you rearrange the order of segments in a flexfield, you produce a different structure. In some applications, different end users need to see individually tailored segment structures for the same flexfield; for example, the correctly formatted local postal address for customer service inquiries, which would change based on the user's locale.

A flexfield can display different segments and prompts for different end users based on a data condition in the application's data, such as the user's role or a value entered by the user. All types of flexfields allow for multiple structures. All the segments that you might need to use to create all the anticipated structures must be defined as part of the flexfield.

• Context: Descriptive and extensible flexfield segments are context-sensitive — they are varied or made available in your application by implementors to reflect the needs of the customer for which the application is being configured. Segments are made available to an application as groups of attributes called contexts. Each context is defined as part of a flexfield, and includes a set of context-sensitive segments that stores a particular type of related information. The database columns on which segments are based can be reused in as many contexts as desired. For example, an implementor can define a Dimensions context, which uses three of the flexfield's database columns of type NUMBER for the height, width, and depth segments. The implementor can also define a Measurements context, which reuses those database columns for weight, volume, and density segments.
Note:
A descriptive flexfield segment that stores data that is applicable to all entities, regardless of context, is referred to as a global segment.

The following links provide more information within this book:

- For more information about the role of implementor, see Participant Roles.
- For more information about developer and customer flexfields, see Descriptive Flexfields and Extensible Flexfields.

Related Links
The following documents provide additional information related to subjects discussed in this section.

For information about flexfield configuration, see the section about defining flexfields in the Implementing Common Features guide for your product family.

For information about implementing your specific product family, do the following:

1. Access the Oracle Fusion Applications library.
2. See the Implementing Common Features guides for your product family.

22.1.1 Descriptive Flexfields

Descriptive flexfields provide a way for customers to add custom attributes to business objects, to define how the attributes are validated, and to define display properties for the attributes. These attributes can be standalone attributes. That is, they do not necessarily need to have anything to do with each other and do not need to be treated as a combination. The segments of a descriptive flexfield that are made available to end users are exposed in the user interface as individual fields.

Descriptive flexfields are entirely optional; customers can choose to configure and expose them or not, as they wish. For example, one customer could configure the parts flexfield to store depth, height, and width, and another customer could configure the parts flexfield to store size and color. Some customers might not need any additional attributes.

You create a descriptive flexfield for one of two purposes:

- For use by customer implementors — a customer flexfield.

  In this case, you define and register the descriptive flexfield in your application database, but all configuration is accomplished by the implementor, including creating contexts, creating segments, and adding validation. End users see these additional attributes in the UI and can enter values for them. End users cannot modify the configuration; they can only enter values for attributes that are already configured.

- To support functionality that you build into your application — a developer flexfield.

  For this purpose, you define and register the descriptive flexfield, and then preconfigure (seed) the contexts and segments, value sets, and validation to satisfy a specific purpose. The descriptive flexfield becomes part of your application, and you can code references to its seeded configuration. You might
also enable implementors to extend the developer flexfield, allowing them to configure it as they would a customer flexfield.

Even if implementors never change a flexfield after it has been configured, they can take advantage of useful flexfield features such as automatic segment validation, automatic segment cross-validation, multiple segment structures, and more.

### 22.1.2 Extensible Flexfields

An extensible flexfield is similar to a descriptive flexfield, but with the added ability for customers to add as many context-sensitive segments to a flexfield as they need. In addition, customers can associate more than one context with any particular row of data.

Extensible flexfields also enable implementors to configure contexts as either single row or multiple row. That is, either one set of the context’s segments is stored for a business object instance, or multiple sets of the context’s segments are stored for the instance. For example, a job position requires only one set of educational requirements, but can require more than one certificate or license.

Extensible flexfields also enable implementors to combine the contexts into groups known as pages, which serve to connect the contexts so they will always be presented together in the application user interface. Hierarchical categories can be defined for extensible flexfields, and implementors can associate any combination of contexts with a given category. For example, the Electronics and Computers category hierarchy might include a Home Entertainment category, which in turn might include an Audio category and a TV category, and so on. The Home Entertainment category might have contexts that specify voltage, dimensions, inputs and outputs. Contexts are reusable within a given extensible flexfield. For example, the dimensions context could be assigned to any category that needs to include dimensional information.

Just as with descriptive flexfields, you create a flexfield for one of two purposes:

- For use by customer implementors — a customer flexfield.
- To support functionality that you build into your application — a developer flexfield.

For more information about customer and developer flexfields, see Descriptive Flexfields.

### 22.1.3 Key Flexfields

Key flexfields are configurable multipart intelligent keys, where each element (segment) of the key may be individually meaningful, as well as the combination as a whole. For example, key flexfields might represent part numbers and account numbers.

A key flexfield is implemented in the user interface as a collection of fields that can be displayed in various UI formats. Key flexfields are never optional; customers must configure them for their Oracle Fusion applications to operate correctly.

The ability to individually tailor key flexfield structures to the end user can be essential. For example, Oracle General Ledger uses a key flexfield called the Accounting Flexfield to uniquely identify a general ledger account. It maintains the multiple accounting codes used throughout Oracle Fusion Applications, and provides different Accounting Flexfield structures for users of different sets of books. Oracle General Ledger determines which flexfield structure to use based on the value of a Set of Books user profile option.
This flexfield is preconfigured to include six segments: company code, cost center, account, product, product line, and subaccount, and valid values are defined for each segment, as well as cross-validation rules to describe valid segment combinations. However, customers might structure their general ledger account fields differently. By including the Accounting Flexfield, Oracle General Ledger can accommodate the needs of different customers. One customer can configure the Accounting Flexfield structure to include 6 segments, while another customer includes 12 segments, all without programming.

22.1.4 Value Sets

An end user enters a value into a flexfield segment while using an Oracle Fusion application. The flexfield validates each segment value against a set of valid values — a value set — which is usually predefined by the implementor. To validate a segment means that the flexfield compares the value that the end user enters into the segment against the values that comprise the value set that is assigned to that segment.

Flexfield segments are usually validated, and typically each segment in a given flexfield uses a different value set. A value set can be assigned to more than one segment and value sets can be shared among different flexfields. For most value sets, when you enter values into a flexfield segment, you can enter only values that already exist in the value set assigned to that segment.

When a business component is generated for a flexfield, the value set metadata is used in the creation of the view objects. Value sets (and their view objects) are owned by a module, which can be an application, a Logical Business Area (LBA), or a sub-LBA in the application taxonomy hierarchy.

For more information about the taxonomy hierarchy, see Working with the Application Taxonomy.

Related Links

The following document provides additional information related to subjects discussed in this section:

For information about value set configuration, see the Define Flexfields section in the Implementing Common Features guide.

For information about implementing your specific product family, do the following:

1. Access the Oracle Fusion Applications library.
2. See the Implementing Common Features guides for your product family.

22.1.5 Flexfield Integration with Oracle Business Intelligence

Oracle Business Intelligence is a comprehensive collection of enterprise business intelligence functionality that provides the full range of business intelligence capabilities including interactive dashboards, proactive intelligence and alerts, enterprise and financial reporting, real-time predictive intelligence, and more.

A flexfield can be included in Oracle Business Intelligence by business-intelligence enabling the flexfield. However, because the polymorphic view objects that are used to model flexfields are not compatible with Oracle Business Intelligence, the flexfield must be flattened into a static form that Oracle Business Intelligence can work with. You accomplish this by slightly modifying the process when you register the flexfield and incorporate it into your application.

For more information, see the following sections in this book:
• Preparing Descriptive Flexfield Business Components for
• Preparing Extensible Flexfield Business Components for
• How to Prepare Key Flexfield Business Components for

22.2 Participant Roles

Responsibility for flexfield development activities is divided between owners and implementors. These are not formal development roles; they are used only in this documentation to clarify and group the flexfield development activities.

Owner
The owner is the developer (or development team) who determines that a particular flexfield is needed or would be useful within a particular Oracle Fusion application, and makes a flexfield of the appropriate design available. The owner then incorporates the flexfield into an application. With key flexfields, the owner can be either a producer or a consumer, or can assume both roles.

• **Producer**: The flexfield producer is the developer who determines that a particular flexfield is needed or would be useful within a particular application, and makes available a flexfield of the appropriate design. With key flexfields, the producer’s product owns the combinations table for that flexfield.

• **Consumer**: A key flexfield consumer incorporates a flexfield into an application. The consumer typically stores a segment value or combination ID (CCID) on a product database table, and works with the structural and seed data and the business components that have been configured by the flexfield producer.

Implementor
An implementor is an individual who sets up all or part of a flexfield-enabled application for deployment. Implementors typically work for or on behalf of customers to install, configure, or administer their applications. In the case of developer flexfields that have been created to support functionality that has been built into the application, the developer also assumes the role of implementor.

For more information about combinations tables for flexfields, see Creating the Combinations Table.

22.3 The Flexfield Development Lifecycle

The process of developing a flexfield and incorporating it into an application differs for each type of flexfield. In general, the process comprises the following activities:

1. Define and register the flexfield metadata.

2. Create business components for the flexfield. For key flexfields, also create business components for the key flexfield combination filters.

3. For descriptive and key flexfields, link the flexfield business components to the application’s business components.

4. Incorporate the flexfield into UI pages.

5. Define sample flexfield data and use it to test the flexfield.
6. Use the Tester role of the Create Flexfield Business Components wizard to create a model that you can use to test the flexfield.

7. Optionally, share the flexfield business components with other developers using an Oracle Application Development Framework (Oracle ADF) library.

After you have completed the flexfield development process and delivered the application, implementors can use the Manage Flexfields task flows to configure each flexfield. These task flows determine how the flexfield’s segments will be populated, organized, and made available to end users within the application.

**Note:**

You must use the specified tools throughout the development lifecycle to create and update the flexfield metadata and product tables. Do not use database tools unless you are instructed to do so. If you use database tools instead of the specified tools, you risk destroying data integrity and you lose the ability to audit changes to the data.

Because Oracle Fusion Applications tables are interrelated, any changes that you make using the specified tools, such as building business components or UI forms, can update many tables at once. If you do not use the specified tools, you might not update all the necessary tables. In addition, most of these tools perform validation and change tracking.

If tables are not properly synchronized, you risk unpredictable results throughout Oracle Fusion Applications products.

For more information about creating business components for key flexfield combination filters, see Working with Code-Combination Filters for Key Flexfields.

**Related Links**

The following document provides additional information related to subjects discussed in this section:

For information about planning and implementing flexfield configuration, such as defining structures, contexts, attributes, labels, behavior, and associated value sets, see the Define Flexfields section in the Implementing Common Features guide.

For information about implementing your specific product family, do the following:

1. Access the Oracle Fusion Applications library.
2. See the Implementing Common Features guides for your product family.

### 22.4 Flexfields in the Application User Interface

Flexfield segments can appear in a user interface as either label/widget pairs or as table fields. After an extensible flexfield is configured, it appears in the UI as one or more regions. Configured descriptive and key flexfields may appear in the UI in either a form layout, a tabular layout, or in a form or table layout in a popup component:

- **Form layout**: A typical label/prompt and either view-only data or a widget (text field, choice list, and so on) that allows an end user to enter values.

- **Tabular layout**: A column of information in a table, where the label of the flexfield segment is the column header, and the values are within each cell of the column.
All segments of a single flexfield are grouped together by default. The layout of the form or table and the positions of the flexfield segments depend on where you place the flexfield on the page. Flexfields may also be presented in a separate section of the page, alone in a table, or on their own page.
This chapter discusses how to use descriptive flexfields to enable customers to add additional attributes to business objects in their Oracle Fusion applications.

This chapter includes the following sections:

- Introduction to Descriptive Flexfields
- Developing Descriptive Flexfields
- Creating Descriptive Flexfield Business Components
- Creating Descriptive Flexfield View Links
- Nesting the Descriptive Flexfield Application Module Instance in the Application Module
- Adding a Descriptive Flexfield View Object to the Application Module
- Adding Descriptive Flexfield UI Components to a Page
- Configuring Descriptive Flexfield UI Components
- Testing Flexfields
- Loading Seed Data
- Working with Descriptive Flexfield UI Programmatically
- Incorporating Descriptive Flexfield into a Search Form
- Preparing Descriptive Flexfield Business Components for
- Publishing Descriptive Flexfields as Web Services
- Accessing Descriptive Flexfields from an ADF Desktop Integration Excel Workbook

23.1 Introduction to Descriptive Flexfields

Descriptive flexfields provide a way for implementors at customer sites to add custom attributes to entities, and to define validation and display properties for them. A descriptive flexfield is a logical grouping of segments that is mapped to a set of database columns that serve as placeholders for custom attributes. These placeholder columns are often referred to as extension columns. The attributes in the group are of three types: global segment, context-sensitive segment, and context segment. The global segments are for custom attributes that apply to all entity rows, while the context-sensitive segments are for custom attributes that apply to certain entity rows based on the value of a context segment.
As the developer, you can define multiple usages for a descriptive flexfield. For example, you might have defined an address flexfield that the implementor may use to add attributes related to addresses. The implementor can define context-sensitive segments for the address that are based on a certain attribute, such as country code. You can reuse the address flexfield with any table for which you need address information, and the implementor needs to configure the flexfield only once.

To complete the development tasks for descriptive flexfields:

1. Create the extension columns to store the flexfield data, and then register the flexfield definition, usage, and parameter metadata.
2. Create descriptive flexfield business components.

   Tip:
   After completing this step, you can regenerate the flexfield business components programmatically at runtime to update your descriptive flexfield implementation without manual intervention.

3. Create view links between the descriptive flexfield business components and the application’s business components.
4. Nest the descriptive flexfield application module instance in the application module.
5. Add an instance of the descriptive flexfield view object to the application module.
6. Add the descriptive flexfield usage to the appropriate application pages.
7. Configure the descriptive flexfield UI components.
8. Test the flexfield.
9. Load any necessary application seed data, such as error messages or lookup values.

After implementing a flexfield, you can define seed or test configurations for the flexfield, and then test it.

After you have completed the flexfield development process and delivered your application, implementors can use the Manage Descriptive Flexfields task flows to define context values and to configure the segments for each flexfield. These task flows determine how the flexfield’s segments will be populated, organized, and made available to end users within the application.

To make the Manage Descriptive Flexfields task flows available to implementors, register them with the Oracle Fusion Functional Setup Manager.

The following links provide more information within this book:

- For more information about flexfield basics and terms, including developer and implementor roles, global segments, context-sensitive segments, and context segments, see Getting Started with Flexfields.
- For more information about developer and implementor roles, see Participant Roles.
- For more information about storing and registering descriptive flexfield data, see Developing Descriptive Flexfields.
• For more information about creating descriptive flexfield business components, see Creating Descriptive Flexfield Business Components.

• For more information about regenerating flexfield business components, see Regenerating Flexfield Business Components Programmatically.

• For more information about creating view links between descriptive flexfield business components, see Creating Descriptive Flexfield View Links.

• For more information about nesting descriptive flexfield application module instances in an application module, see Nesting the Descriptive Flexfield Application Module Instance in the Application Module.

• For more information about adding descriptive flexfield view objects to an application module, see Adding a Descriptive Flexfield View Object to the Application Module.

• For more information about adding descriptive flexfield components to application pages, see Adding Descriptive Flexfield UI Components to a Page.

• For more information about configuring descriptive flexfield UI components, see Configuring Descriptive Flexfield UI Components.

• For more information about testing flexfields, see Testing Flexfields.

• For more information about loading application seed data, see Loading Seed Data.

• For more information about testing and deploying flexfields, see Introduction to Testing and Deploying Flexfields.

• For more information about registering descriptive flexfield task flows, see Integrating Flexfield Task Flows into.

Related Links

The following documents provide additional information related to subjects discussed in this section:

For information about implementing your specific product family, do the following:

• Access the Oracle Fusion Applications Technology library.

• See the Implementing Common Features guides for your product family. Search for flexfields.

23.1.1 Benefits of Descriptive Flexfields

Descriptive flexfields let you satisfy different customers without having to reprogram the application, by enabling the customers to add customized fields. Descriptive flexfields also enable context-sensitive fields that appear only when needed. In essence, a descriptive flexfield enables implementors to extend the data model without writing either XML or Java. A descriptive flexfield is presented as a set of fields on a page, much like the fields of the core application.

For example, consider a retail application that keeps track of customers. The customer form would typically include fields such as Name, Address, State, Customer Number, and so on. However, the page might not include fields to keep track of customer clothing size and color preferences, because these are attributes that can differ for each company that uses the application. For example, if the retail application is used for a tool company, a field for clothing size would be undesirable. Even if you initially
provide all the fields that a company needs, the company might later identify even more customer attributes that it wants to track. You can add a descriptive flexfield to the customer form to provide the desired expansion space. Companies also can take advantage of the fact that descriptive flexfields can be context-sensitive, where the information that the application stores depends on other values that the end users enter on other parts of the page. For example, a company could configure the descriptive flexfield for a fixed asset form to store style, size, and wood type if the asset type was "desk", and store CPU chip and memory size if the asset type was "computer."

Another example is the descriptive flexfield in the Oracle General Ledger journal entry form. Implementors can configure the flexfield to add information of a customer's own choosing. For example, a customer might want to capture additional information about each journal entry, such as source document number, or the name of the person who prepared the entry.

To maximize flexibility for customers, consider defining a descriptive flexfield for every entity in your application to which a customer might need to add attributes.

23.1.2 How Descriptive Flexfields Are Modeled in Oracle Application Development Framework

Flexfields are modeled as a collection of Oracle Application Development Framework (Oracle ADF) polymorphic view rows, as described in the Working with Polymorphic View Rows section in the Developing Fusion Web Applications with Oracle Application Development Framework. In a polymorphic collection of rows, each view row can have its own set of attributes, and all rows have at least one common attribute, the discriminator. The discriminator determines which view row type is used. Given a collection of polymorphic view rows, each row can be a different type.

The attribute sets that are associated with the discriminator are predefined. In fact, Oracle ADF enables each view row to have its own view definition. When a polymorphic collection of rows is created, Oracle ADF selects a view definition for the row to be added based on the value of the discriminator attribute.

Descriptive flexfield segments are exposed as view row attributes in the order that they are defined in the flexfield's metadata. Global segments are exposed as attributes in the base view object of the polymorphic collection. Every context is modeled as an extended view object of the base view object. That is, an extended view object is created for every context value. These extended view objects, which are referred to as subtype view objects, expose context-sensitive segments as subtype-specific view attributes. The context segment is exposed as the discriminator attribute of the polymorphic view rows.

You use a wizard to generate a polymorphic base view object that is based on the descriptive flexfield definition, then create a view link to connect the product view object and the base view object. You can then use the base view object to add the flexfield to a UI page. For more information about the generation of base and subtype view objects, see Creating Descriptive Flexfield Business Components.
Note:

One distinction of the flexfield context-switching mechanism is that during context switching, the context-sensitive segments are initialized as follows:

- If a segment is defined to use a constant value from an underlying entity object as its default, it is initialized to that value.
- If a segment is defined to use a descriptive flexfield parameter by default, it is initialized to that value.
- For all other cases, the context-sensitive segment is set to NULL.

For more information about polymorphic view rows, see the Working with Polymorphic View Rows section in the Developing Fusion Web Applications with Oracle Application Development Framework.

Note:

Because flexfield view objects are modeled as polymorphic view objects, you can use descriptive flexfield view objects in the same manner that you use any other polymorphic view objects, and they will behave in the same way. This includes support for flexfields in ADF Desktop Integration. For more information, see Accessing Descriptive Flexfields from an ADF Desktop Integration Excel Workbook and the Oracle Fusion Middleware Developing Applications with Oracle ADF Desktop Integration guide.

23.2 Develop Descriptive Flexfields

Whenever you have a product table that you think implementors might need to extend for their specific circumstances, you can add columns to the table and register those columns as flexfield segments. After you have registered a flexfield, you can reuse the flexfield with other product tables.

To complete the process for developing a descriptive flexfield:

1. Add extension columns to the product table.
2. Register the flexfield and define its metadata and primary usage.
3. Register the entity details and parameters for the primary usage.
4. Optionally reuse the flexfield by adding the same set of extension columns to other product tables.
5. Register the secondary usages and their entity details.

23.2.1 How to Create Descriptive Flexfield Columns

To implement a descriptive flexfield for a product table, you first add extension columns to that table. You need to add a context column, such as ATTRIBUTECATEGORY, and as many generic attribute (segment) columns of each type, such as ATTRIBUTECHAR1 and ATTRIBUTENUMBER12, as will be needed by the implementors. A segment column must be a VARCHAR2, NUMBER, DATE, or
When using a flexfield to add a custom attribute, the implementor maps the custom attribute to an available extension column.

**Tip:**
There are no constraints on how to name the segment columns. However, these columns are typically named using the patterns ATTRIBUTE_CHAR\textsubscript{n}, ATTRIBUTE_NUMBER\textsubscript{n}, ATTRIBUTE_DATE\textsubscript{n}, and ATTRIBUTE_TIMESTAMP\textsubscript{n}. This convention makes it easy to identify the flexfield segments. It also makes it easier to name the columns for other usages of the flexfield.

The context column, which is required, must be of type VARCHAR2. The context column’s length determines the maximum length of the context codes that can be created by implementors when they configure the flexfields.

Each implementor can configure as many of the segment columns as the end user requires and can choose whether to use the context column.

You must use the Database Schema Deployment Framework tools to create the product table and columns. Using these tools ensures that the table and its columns are registered in the Oracle Fusion Middleware Extensions for Applications (Applications Core) data dictionary. For more information, see Using the Database Schema Deployment Framework.

### 23.2.2 How to Register and Define Descriptive Flexfields

Before you can create business components for a descriptive flexfield, you must first define and register the descriptive flexfield.

The basic steps for defining and registering a descriptive flexfield are as follows:

- Name and describe the flexfield.
- Indicate whether the flexfield should be protected from end-user configuration. If a flexfield is protected, it cannot be edited using setup tasks, such as Manage Descriptive Flexfields. Typically, this option is used for developer flexfields. Note that you must use the APIs to do this step. For information, see Using the Setup APIs To Register and Define Descriptive Flexfields.
- Define the primary usage. The primary usage is the first usage that you define for a flexfield.
- Define the context column to be used for the context segment. The product table that is used for the primary usage is called the primary table.
- Define the product table columns to be used for the flexfield segments.

There are two ways in which you can define a descriptive flexfield:

- Using the Flexfield Registration Metadata wizard. If you use the wizard, you also define the entity details for the primary usage and you define the parameters.
- Using the setup APIs in the FND_FLEX_DF_SETUP_APIS PL/SQL package.

#### 23.2.2.1 Using the Flexfield Registration Metadata Wizard To Register and Define Descriptive Flexfields

You can use the Register new flexfield operation from the Flexfield Registration Metadata wizard to register and define a descriptive flexfield.

Develop Descriptive Flexfields
Tip:
To make subsequent flexfield additions and changes, you can use the Edit flexfield operation from the Flexfield Registration Metadata wizard.

Before you begin:

- Create the extension columns as described in How to Create Descriptive Flexfield Columns.
- Ensure that entity objects exist for the primary table.

To register and define a descriptive flexfield using the Flexfield Registration Metadata wizard:

1. From the File menu, choose New.
2. In the New Gallery, expand Business Tier, select ADF Business Components and then Flexfield Registration Metadata, and click OK.
3. On the Choose Operation page of the Flexfield Registration Metadata wizard, select Register new flexfield as shown in Figure 23-1, and then click Next.

   **Figure 23-1  Flexfield Registration Metadata Wizard — Choose Operation Page**

4. On the Basic Information page, select Descriptive Flexfield from the Type dropdown list, as shown in Figure 23-2.
5. Provide the following values:

- **Code**: Type a code that uniquely identifies the flexfield.
- **Name**: Type a descriptive name for the flexfield.
- **Description**: Type a short description of the flexfield.
- **Application**: Select the application name that has been assigned for the product area.
- **Module**: Select the module that owns the flexfield. This is typically the application or logical business area (LBA) with which the flexfield is delivered.
- **Delimiter**: Select the character to be displayed between flexfield segments, when the segments are displayed in a concatenated format.
- **Enable Business Intelligence**: (Optional) Select the checkbox to enable the flexfield for Oracle Business Intelligence. For more information, see Preparing Descriptive Flexfield Business Components for.

6. Click Next.

7. On the Primary Table page, set the following values to define the primary usage for the flexfield, as shown in Figure 23-3.

- **Base Table**: Enter or select the name of the database table that contains the columns to be used as flexfield segments.
- **Table Usage Code**: Type a code that uniquely identifies the flexfield usage. For the primary usage, this is typically the same code as the flexfield code.
- **Table Usage Name**: Type a descriptive name for the flexfield usage.
8. From the **Context Column** dropdown list, select the database column that you want to map to the flexfield’s context segment. The column must be of type VARCHAR2.

**Tip:**

The context segment database column is typically named ATTRIBUTE_CATEGORY.

9. In the **Available** list in the **Segment Columns** region, select the database columns that you want to map to flexfield segments and move them to the **Selected** list.

10. Click **Next**.

11. On the Entities page, select the entity objects for the table upon which the usage is based and move them to the **Selected** list, as shown in **Figure 23-4**.
12. Select each entity object from the **Selected** list and provide the following values:

- **Object Name Prefix**: Enter a short unique name for the flexfield usage. For example, `PartsDF`. This prefix is used to derive the names of objects that are generated for the flexfield usage.

- **Package Name**: Specify the name of the root package to be used for the generated business components that model the flexfield usage.

- **BI Flattened Fact Name**: If the flexfield is enabled for Oracle Business Intelligence and you know the Oracle Business Intelligence object that this flexfield usage should be mapped to when the flexfield is imported into Oracle Business Intelligence, specify the name of the object.

  For more information about Oracle Business Intelligence, see *Preparing Descriptive Flexfield Business Components for*.

13. Click **Next**.

14. On the Parameters Page, optionally define parameters for the entity object attributes that can be used to pass external reference data to flexfield segments, as shown in *Figure 23-5*. 
15. On the Summary page, optionally select Extract seed data for the flexfield to create a seed data file for uploading the registration information into other implementations. For more information, see Loading Seed Data.

16. If you are ready to create business components for the flexfield, select Launch Flexfield Business Components Wizard.

17. Click Finish.

23.2.2.2 Using the Setup APIs To Register and Define Descriptive Flexfields

You can define and register a descriptive flexfield using procedures from the FND_FLEX_DF_SETUP_APIS PL/SQL package. This package also has procedures for updating, deleting, and querying flexfield definitions.

To learn how to access documentation about using the FND_FLEX_DF_SETUP_APIS PL/SQL package, see What You May Need to Know About the Descriptive Flexfield Setup API.

Before you begin:
Create the extension columns as described in How to Create Descriptive Flexfield Columns.

To register and define a descriptive flexfield using the setup APIs:

1. Run the fnd_flex_df_setup_apis.create_flexfield(...) procedure to register the descriptive flexfield, its context segment, and its primary (master) usage. If you do not want end users to configure the flexfield, set the protected flag to N.

2. Run the fnd_flex_df_setup_apis.create_segment_column_usage(...) procedure for each segment column to register the segment columns.
3. (Optionally) Register the entity details as described in How to Register Entity Details. This step must be completed before you can generate the flexfield usage's business components.

23.2.2.2.1 What You May Need to Know About the Descriptive Flexfield Setup API

In the descriptive flexfield development process, use the FND_FLEX_DF_SETUP_APIS PL/SQL package to manage flexfield registration metadata.

You can learn about the FND_FLEX_DF_SETUP_APIS PL/SQL package by running the following command, which stores package documentation and usage examples in the <db_name>_<user_name>_FND_FLEX_DF_SETUP_APIS_<date>.plsqldoc file.

```
sqlplus <fusion_user>/<fusion_pwd>@<fusion_db> @/ORACLE/fusionapps/atgpf/applcore/db/sql/flex/fnd_flex_pkg_doc.sql FND_FLEX_DF_SETUP_APIS
```

23.2.3 How to Reuse a Descriptive Flexfield on Another Table

A descriptive flexfield configuration can be shared with other product tables. To reuse a descriptive flexfield, you add the same set of extension columns to the product table for which you want to reuse the flexfield, and you register the secondary usage as described in How to Register the Reuse of a Descriptive Flexfield.

The product table that was used to register the flexfield is referred to as the primary table, and it is the owner of the flexfield. The usage that you create when you register the flexfield is called the primary usage. A reuse of a flexfield is referred to as a secondary usage.

The secondary table must have the same number of extension columns as the primary table. The secondary extension columns must have the same data type and size as the corresponding primary table extension columns.

The column names must also be the same as in the primary table, except for an optional prefix. For example, if the column names are ATTRIBUTE1 and ATTRIBUTE2 in the primary table, then in the secondary table they could again be ATTRIBUTE1 and ATTRIBUTE2, respectively, or with a prefix, they could be HOME_ATTRIBUTE1 and HOME_ATTRIBUTE2. They cannot be ATTR1, HOME_ATTR2, or any variation that does not end in the names of the primary table columns.

When configuring the flexfield, the implementor can configure as many of the segment columns as the end user requires and can choose whether to use the context column.

You must use the Database Schema Deployment Framework tools to create the product table and columns. Using these tools ensures that the table and its columns are registered in the Oracle Fusion Middleware Extensions for Applications (Applications Core) data dictionary. For more information, see Using the Database Schema Deployment Framework.

23.2.4 How to Register the Reuse of a Descriptive Flexfield

After you add extension columns to a table for a secondary usage of a flexfield, you must register the usage before you can build a descriptive business component for the secondary usage.

There are two ways in which you can register a secondary usage:
23.2.4.1 Using the Flexfield Registration Metadata Wizard To Register a Secondary Usage

You can use the **Register new flexfield usage operation** from the Flexfield Registration Metadata wizard to register the secondary usage of a descriptive flexfield. The registration wizard uses the primary usage column mappings to determine how to map the secondary table’s column names to the flexfield segments. If you specify a prefix, the wizard uses the prefix to determine the column mappings. For example, if the primary table’s ATTRIBUTE1 column is mapped to a segment, and you specify a prefix of HOME for the secondary usage, the wizard automatically maps its HOME_ATTRIBUTE1 column to a flexfield segment.

Before you begin:

1. Register the descriptive flexfield and its primary usage as described in *How to Register and Define Descriptive Flexfields*.
2. Create the extension columns for the secondary usage as described in *How to Reuse a Descriptive Flexfield on Another Table*.
3. Ensure that entity objects exist for the secondary table.

**To register a second usage using the Flexfield Registration Metadata wizard:**

1. From the **File** menu, choose **New**.
2. In the New Gallery, expand **Business Tier**, select **ADF Business Components** and then **Flexfield Business Components**, and click **OK**.
3. On the Choose Operation page of the Flexfield Registration Metadata wizard, select **Register new flexfield usage**, and then click **Next**.
4. On the Select Usage page, select **Descriptive** from the **Type** dropdown list, as shown in Figure 23-6.
5. To display available usages, provide a value for Application, Code, or Table. You can use a wildcard character, such as a percent sign (%), to match on similar values.

6. Select the primary usage of the flexfield for which you want to create the secondary usage, and then click Next.

7. On the Basic Information page, review the information about the primary usage, and then click Next.

8. On the Secondary Table page, set the following values to define the secondary usage for the flexfield, as shown in Figure 23-7.

   - **Base Table**: Enter or select the name of the secondary table.
   - **Table Usage Code**: Type a code that uniquely identifies the secondary usage.
   - **Table Usage Name**: Type a descriptive name for the flexfield usage.
   - **Column Name Prefix**: If you used a prefix for the flexfield column names, enter the prefix.
9. Click Next.

10. On the Entities page, select the entity objects for the secondary table and move them to the **Selected** list.

11. Select each entity object from the **Selected** list and provide the following values:

- **Object Name Prefix**: Enter a short unique name for the secondary usage. For example, `PartsDFFIntf`. This prefix is used to derive the names of objects that are generated for the flexfield usage.

- **Package Name**: Specify the name of the root package to be used for the generated business components that model the secondary usage.

**Note:**

Each usage must have a unique package name. In addition, the package name must uniquely identify a usage. For example, if the root package for a usage is `oracle.apps.flex.hcm.payroll.dff1`, then you cannot define the `oracle.apps.flex.hcm.payroll` package for another usage, because that package would then identify both usages. Instead, you could use `oracle.apps.flex.hcm.payroll.dff2`.

- **BI Flattened Fact Name**: If the flexfield is enabled for Oracle Business Intelligence and you know the Oracle Business Intelligence object that this usage should be mapped to when the flexfield is imported into Oracle Business Intelligence, specify the name of the object.

For more information about Oracle Business Intelligence, see [Preparing Descriptive Flexfield Business Components](#) for...
12. Click Next.

13. On the Summary page, optionally select Extract seed data for the flexfield to create a seed data file for uploading the registration information into other implementations. For more information, see Loading Seed Data.

14. If you are ready to create business components for the flexfield, select Launch Flexfield Business Components Wizard.

15. Click Finish.

23.2.4.2 Using the Setup APIs To Register a Secondary Usage

You can define and register the secondary usage of a descriptive flexfield using procedures from the `FND_FLEX_DF_SETUP_API` PL/SQL package.

To learn how to generate documentation about using the `FND_FLEX_DF_SETUP_API` PL/SQL package, see What You May Need to Know About the Descriptive Flexfield Setup API.

The definition of a descriptive flexfield usage includes the following information:

- The secondary table name
- The code for identifying the secondary usage
- The column name prefix. For example, if the primary table column is `ATTRIBUTE1`, and the secondary table column is `HOME_ATTRIBUTE1`, then the prefix is `HOME_`.

Before you begin:

1. Register the descriptive flexfield and its primary usage as described in How to Register and Define Descriptive Flexfields.

2. Create the extension columns for the secondary usage as described in How to Reuse a Descriptive Flexfield on Another Table.

To register a secondary usage using the setup APIs:

1. To register the secondary usage of a descriptive flexfield, run the `fnd_flex_df_setup_apis.create_flex_table_usage(...)` procedure.

2. (Optionally) Register the entity details as described in How to Register Entity Details. This step must be completed before you can generate the flexfield usage's business components.

23.2.5 How to Register Entity Details

Before you can create a flexfield business component and create flexfield-specific application module instances, you must register the following information:

- The full class name of the entity object for the table upon which the flexfield usage is based.
- A prefix from which to derive the names of generated objects.
- The package in which to place the generated business components. Each usage must have its own package name.
Note:
Each usage must have a unique package name. In addition, the package name must uniquely identify a usage. For example, if the root package for a usage is `oracle.apps.flex.hcm.payroll.dff1`, then you cannot define the `oracle.apps.flex.hcm.payroll` package for another usage, because that package would then identify both usages. Instead, you could use `oracle.apps.flex.hcm.payroll.dff2`.

If you used the Flexfield Registration Metadata wizard to define the flexfield usage, then you have already registered the entity details. Otherwise, you can use the Flexfield Registration Metadata wizard or the `FND_FLEX_DF_SETUP_APIS` PL/SQL package to register this information for a flexfield usage.

23.2.5.1 Using the Flexfield Registration Metadata Wizard to Register Entity Details
You can use the Edit flexfield usage operation from the Flexfield Registration Metadata Wizard to register entity details.

To register a usage using the Flexfield Registration Metadata wizard:

1. From the File menu, choose New.
2. In the New Gallery, expand Business Tier, select ADF Business Components and then Flexfield Business Components, and click OK.
3. On the Choose Operation page of the Flexfield Registration Metadata wizard, select Edit Flexfield Usage, and then click Next.
4. To display available usages, provide a value for Application, Code, or Table. You can use a wildcard character, such as a percent sign (%), to match on similar values.
5. Select the flexfield usage for which you want to add entity details.
6. Click Next until you are on the Entities page.
7. On the Entities page, select the entity objects for the table upon which the usage is based and move them to the Selected list.
8. Select each newly added entity object from the Selected list and provide the following values:
   - **Object Name Prefix**: Enter a short unique name for the flexfield usage. For example PartsDFF. This prefix is used to derive the names of objects that are generated for the flexfield usage.
   - **Package Name**: Specify the name of the root package to be used for the generated business components that model the flexfield usage.
   - **BI Flattened Fact Name**: If the flexfield is enabled for Oracle Business Intelligence and you know the Oracle Business Intelligence object that this flexfield should be mapped to when the flexfield is imported into Oracle Business Intelligence, specify the name of the object.

For more information about Oracle Business Intelligence, see Preparing Descriptive Flexfield Business Components for.
9. Click Next, and click Next again.

10. On the Summary page, optionally select Extract seed data for the flexfield to create a seed data file for uploading the registration information into other implementations. For more information, see Loading Seed Data.

11. If you are ready to create business components for the flexfield, select Launch Flexfield Business Components Wizard.

12. Click Finish.

23.2.5.2 Using the Setup APIs To Register Entity Details

You can register entity details using procedures from the FND_FLEX_DF_SETUP_APIS PL/SQL package.

To learn how to generate documentation about using the FND_FLEX_DF_SETUP_APIS PL/SQL package, see What You May Need to Know About the Descriptive Flexfield Setup API.

Before you begin:

1. Register the usage as described in How to Register and Define Descriptive Flexfields and How to Register the Reuse of a Descriptive Flexfield.

2. Ensure that the entity object for the usage’s table exists.

To register the entity details using the setup APIs:

• Run the fnd_flex_df_setup.Apis.create_adfbc_usage(...) procedure to register the entity object, package name, and object name prefix for the flexfield usage.

23.2.6 How to Register Extensible Flexfield Parameters

A flexfield parameter is a declared public variable, which can be used to designate which attributes of eligible entity objects that are related to the flexfield can be used to pass external reference data to flexfield segments. These entity object attributes could, in turn, take their values from column values, constant values, session attributes, and so forth.

A flexfield may have zero, one, or many flexfield parameters defined, each one representing a specific type of information that is useful to that flexfield. Implementors can use the parameters to define defaults and value set validation for the flexfield segments.

Some or all of these types of data sources can be referenced in the WHERE clauses for table value sets.

in the following ways:

• At row creation time to provide default segment values.

• In derived segments, which are global, context, or context-sensitive segments for which values are derived from an external reference data source, causing its values to automatically change to reflect any new reference data values.

• In the WHERE clauses for table value sets.
Note:
Although a flexfield parameter is associated with a flexfield in metadata, it is not connected with any specific segment in the flexfield. Rather, it serves as a variable through which flexfield segments can access reference data from other sources.

Every flexfield parameter must be mapped to an appropriate entity object attribute at design time. In this way, implementors are guaranteed that the parameters will always be mapped to entity object attributes, and they can use the parameters as needed.

When you create business components for a descriptive flexfield, you will be required to map each parameter associated with that flexfield to an attribute of the entity object that you are creating. The values accessed from reference data sources by these parameters are then available for you to use in your application. Many of the core (nonflexfield) fields on a page can serve as reference fields.

Consider the example of an Expense Lines entity object with the core fields of Expense Line ID, Expense Date, Amount, Description, and Expense Type. If the flexfield has an ExpenseType parameter that is mapped to the Expense Type field, an implementor can configure the context segment to derive its value from the ExpenseType parameter.

To implement descriptive flexfield parameters, you must map them to the appropriate entity object attributes at design time and configure them. For more information, see Creating Descriptive Flexfield Business Components and How to Configure Descriptive Flexfield Parameters.

To provide flexfield parameter values at runtime, the base implementation of the context entity object must override the method `getFlexfieldParameterValue` of `oracle.apps.fnd.appcore.oaext.model.OAEntityImpl`. You must handle every flexfield parameter in the overriding method. This guarantees that the parameters always contain the most up-to-date information.

The overriding method must return an object of appropriate type based on the flexfield parameter data type, as shown in the following table.

<table>
<thead>
<tr>
<th>Flexfield Parameter Type</th>
<th>Return Data Object Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>Number</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>Date</td>
<td>java.sql.Date</td>
</tr>
<tr>
<td>Timestamp</td>
<td>java.sql.Timestamp</td>
</tr>
</tbody>
</table>

The following code sample shows how to provide flexfield parameter values at runtime by overriding the `getFlexfieldParameterValue` method of the context entity object.

Example 23-1 Providing flexfield parameter values at runtime

```
public class BaseContextEOImpl extends OAEntityImpl {  
    @Override 
    public Object getFlexfieldParameterValue(final String localFlexID, final String paramCode) { 
        if ("EXPENSE_TYPE".equals(paramCode)) { 
            return 
```
getAttribute("ExpenseType");  }  else if ("EXPENSE_AMOUNT".equals(paramCode))
{    Double amount = (Double) getAttribute("ExpenseAmount");    return (amount
== null ? null : new java.math.BigDecimal(amount));  }  else
{    return super.getFlexfieldParameterValue(localFlexID, paramCode);  }
}

If you used the Flexfield Registration Metadata wizard to define and register the
flexfield, then you might have already defined the parameters. Otherwise, you can
edit the flexfield using the Flexfield Registration Metadata wizard or the setup APIs.

23.2.6.1 Use the Flexfield Registration Metadata Wizard to Register a Flexfield
Parameter

You can use the Edit flexfield usage operation from the Flexfield Registration
Metadata Wizard to register flexfield parameters.

To register a parameter using the Flexfield Registration Metadata wizard:

1. From the File menu, choose New.
2. In the New Gallery, expand Business Tier, select ADF Business Components and
then Flexfield Business Components, and click OK.
3. On the Choose Operation page of the Flexfield Registration Metadata wizard,
select Edit Flexfield Usage, and then click Next.
4. To display available usages, provide a value for Application, Code, or Table. You
can use a wildcard character, such as a percent sign (%), to match on similar
values.
5. Select the primary usage of the flexfield for which you want to add the
parameters.
6. Click Next until you are on the Parameters page.
7. On the Parameters Page, optionally define parameters for the entity object
attributes that can be used to pass external reference data to flexfield segments.
8. Click Next.
9. On the Summary page, optionally select Extract seed data for the flexfield to
create a seed data file for uploading the registration information into other
implementations. For more information, see Loading Seed Data.
10. If you are ready to create business components for the flexfield, select Launch
Flexfield Business Components Wizard.
11. Click Finish.

23.2.6.2 Register a Flexfield Parameter Using the Setup APIs

You can register a flexfield parameter using procedures from the
FND_FLEX_EF_SETUP_APIS PL/SQL package.

To learn how to generate documentation about using the
FND_FLEX_EF_SETUP_APIS PL/SQL package, see What You May Need to Know
About the Descriptive Flexfield Setup API.

Before you begin:
Define and register the flexfield as described in How to Register and Define Descriptive Flexfields.

To create a parameter using the setup APIs:

- Run the `fnd_flex_ef_setup_apis.create_parameter(...)` procedure and provide a parameter code and data type.

### 23.3 Create Descriptive Flexfield Business Components

Before you can use a descriptive flexfield in your application, you must use the Create Flexfield Business Components wizard to generate flexfield business components for the flexfield. The wizard generates a base view object that is based on the information in the flexfield metadata. After the initial flexfield registration, and before any configuration is completed, the base view object has at least two attributes: the primary key attribute, which links the flexfield view object to the product view object, and the context attribute, which serves as the discriminator.

When implementors configure the flexfields by defining global and context-sensitive segments, the base view object is regenerated and additional flexfield view objects are generated. Figure 23-8 shows an example of the configured ADF Business Components. The product view object contains only the nonflexfield attributes. The base view object contains the primary key attribute, the context attribute, and global attributes. The base view object is extended to define view object rows based on the configured context values. Each context value requires a subtype view object definition that represents the structure of the rows with that context value, as described in How Descriptive Flexfields Are Modeled in.

**Figure 23-8  Descriptive Flexfield Modeled as ADF Business Components Component**
Note:
The product view object might contain other attributes. However, the product view object must not include flexfield view object attributes.
None of the flexfield view objects contains the fixed (nonflexfield) columns.

No Java implementation classes are generated for descriptive flexfield view objects. The product view object may or may not have Java implementation classes.

23.3.1 How to Create Descriptive Flexfield Business Components

You use the Create Flexfield Business Components wizard to create the flexfield business components for a flexfield’s usage.
The business components generated will replace any existing ones that are based on the same flexfield usage.

Before you begin:

• Ensure that you have added the Applications Core library to your project as described in Setting Up Your JDeveloper Application Workspace and Projects.

• Ensure that at least one customization class is included in the adf-config.xml file. This inclusion serves to ensure correct application behavior. It does not matter which customization class you include.

For information about customization layers, see the "Understanding Customization Layers" section in the Extensibility Guide for Developers.

• Verify that the entity object that will be used as the data source for the business component meets the following requirements:
  – All flexfield columns are included in the entity object. In general, all columns should be included.
  – A primary key is defined for the entity object. If an entity object is going to be used to create new application transaction rows, a default primary key must be programmed.
  – All VARCHAR2 columns used for descriptive flexfield attributes are mapped to data type java.lang.String.
  – All number columns used for descriptive flexfield attributes are mapped to data type java.math.BigDecimal.
  – All date columns used for descriptive flexfield attributes are mapped to data type java.sql.Date.
  – All timestamp columns used for descriptive flexfield attributes are mapped to data type java.sql.Timestamp.

• Register the flexfield usage’s entity object, package name, and object name prefix. For more information, see How to Register Entity Details.

To create descriptive flexfield business components:

1. Build your project to ensure that the entity objects are available in classes. The modeler relies on what is in your classes.
2. From the File menu, choose New.

3. In the New Gallery, expand Business Tier, select ADF Business Components and then Flexfield Business Components, and click OK.

4. On the Flexfield page of the Create Flexfield Business Components wizard, select Descriptive from the Type dropdown list as shown in Figure 23-9.

Figure 23-9 Create Flexfield Business Components Wizard — Flexfield Page

5. In the Application field, specify the full name of the application to which your descriptive flexfield belongs.

You can browse for the name, and filter by ID, Short Name, or Name.

6. In the Code field, specify the code of the descriptive flexfield you want to use.

You can browse for and filter by Code.

7. In the Usage section, select the table row that contains your desired descriptive flexfield usage. The descriptive flexfield usage can be one of two possible types:

- The primary usage of the descriptive flexfield on the product table where it was originally defined. Every descriptive flexfield has one primary usage.

- A secondary usage of the descriptive flexfield on a product table, including the one on which it was originally defined. Zero or more secondary usage instances can be defined for a given flexfield, each one potentially on a different product table. You can identify secondary usage instances by the presence of the prefix (Reuse) in the Description field.

8. Click Next.

9. On the Path page, select the path of the directory structure in which to create the flexfield business components and click Next.

10. On the Entity Object page, select the entity object to use as the data source for the descriptive flexfield, as shown in Figure 23-10.
The entity object you select must include all of the attributes representing the columns that are reserved for the descriptive flexfield.

**Note:**

If you select a polymorphic entity object, ensure that the `InheritPersonalization` property for every subtype entity is set to `true`.

Descriptive flexfield attributes will be validated by the entity object along with its other attributes.

11. If the entity object for the descriptive flexfield has attributes that are defined as transient (not based on database table columns), then select **The names of the flexfield attributes match the names of registered columns exactly; do not check the underlying columns.**

When an attribute of a descriptive flexfield entity object is transient, there is no matching underlying column name. When you select this option, the system will match the entity object attribute names to the descriptive flexfield column names, and use the matching attributes to access the flexfield data. Ensure that the entity object has a full set of attributes with matching names before you select this option.

This entity object must be registered under the primary usage. There is no need to register another table for this purpose, even if the entity object is based on some other table. For more information, see **How to Register Entity Details.**
Note:
If the entity object with transient descriptive flexfield attributes is not based on
the primary usage, the transient attributes must be named using the same
prefix as the other attributes of that entity object (and the corresponding table
columns). For more information, see How to Reuse a Descriptive Flexfield on
Another Table.

Caution:
The Create Flexfield Business Components wizard is case-sensitive. All
column names — and the names of the flexfield entity object attributes
associated with them — must be uppercase.

12. Click Next.

13. The Naming page displays the entity object’s package name and object name that
you registered for the usage, as described in How to Register Entity Details. Click
Next.

14. On the Parameters page shown in Figure 23-11, map each flexfield parameter to the
entity object attribute that will be the data source for the parameter.

Parameters are not a requirement for descriptive flexfields. If no parameters are
defined for the descriptive flexfield that you are working with, the Parameters page
will display a message to that effect. However, if any parameters are defined and
associated with a descriptive flexfield, you must map each parameter to an entity
object attribute before you can use the flexfield in your application.

Figure 23-11 Create Flexfield Business Components Wizard — Parameters Page

The names in the Parameter Code column represent parameters that have been
defined for the descriptive flexfield. For each parameter listed, select the entity
object attribute from the Entity Attribute Name dropdown list to use as the data
source for that parameter.
The entity attributes in the dropdown list include the following:

- Attributes that are part of the entity object you selected as the flexfield data source.
- Attributes that are part of any entity object that is directly associated with the flexfield entity object through an accessor.
- Attributes that are part of any entity object that is indirectly associated with the flexfield entity object through a chain of accessors and entity objects.

**Note:**

An entity attribute is available on this list only if all the accessors in the chain to the attribute have an underlying association cardinality of 1-to-1 or many-to-1.

The path through the chain of accessors to each available entity attribute is displayed using the following notation:

`[accessorname1.[accessorname2.]...]attributename`

Although it is not visible, the name of the previously selected flexfield entity object is implied as the first element in the chain, followed by zero or more accessor names, then the target entity attribute name. The names of the entity objects in this chain are also implied.

**Caution:**

- Flexfield parameters can be used only with segments of the same Java type. The data type of each entity attribute you select must match the data type shown for the parameter.

- For any flexfield segment value that is derived from a parameter, the derivation happens only when the value of the mapped attribute is updated. Therefore, if you intend for customers to use a parameter for derivation, then the attribute that you map the parameter to must be updatable. If the parameter’s value must come from a reference-only entity, then create an updatable transient entity attribute and assign that entity to the flexfield parameter.

15. When all of the defined parameters are mapped, click **Next**.

16. On the Summary page, click **Finish**.

**Note:**

This wizard might fail with a "ClassNotFound" exception message. This indicates that one or more required libraries have not been automatically included in your project. You can resolve this issue by manually adding any missing libraries; then you can complete this procedure successfully.
17. Refresh the project to see the newly created flexfield business components in the Application Navigator. The components are in the package that you registered for the flexfield usage and are named using the registered prefix.

23.4 Create Descriptive Flexfield View Links

A view link is needed whenever a product view object references your descriptive flexfield. The product view object and the flexfield base view object are linked through their primary keys.

23.4.1 How to Create Descriptive Flexfield View Links

You create a view link to connect your product view object with the flexfield view object. After you have created the view link, you can use the view object to add the flexfield to an application page.

Before you begin:

1. Create the master view object, which contains only nonflexfield attributes.
2. Create the flexfield business components for the descriptive flexfield as described in How to Create Descriptive Flexfield Business Components.

To create a descriptive flexfield view link:

1. From the File menu, choose New.
2. In the New Gallery, go to Business Tier > ADF Business Components and select Flexfield View Link.
3. Click OK to access the Create Flexfield View Link wizard, as shown in Figure 23-12.

Figure 23-12 Create Flexfield View Link Wizard — Name Page

4. On the Name page, from the Package dropdown list, specify a package for the view link.
Caution:
You cannot move the view link to a different package after you create it. Instead, you must delete the view link and re-create it with the new package name.

5. In the Name field, enter a name for the view link.

6. Click Next. The View Objects page appears, as shown in Figure 23-13.

**Figure 23-13   Create Flexfield View Link Wizard — View Objects Page**

7. In the Select Source View Object tree, expand the available objects from your current project and select the master view object.

8. In the Select Destination Flexfield tree, expand the application and select a destination flexfield usage.

9. In the View Link Accessor Name field, enter an appropriate name for the view link accessor.

10. Click Next.


For descriptive flexfields, the Source Attributes page is informational only. The wizard uses the primary key attributes of the source view object to define the view link.

**Note:**
You can skip the Properties page because view link-specific properties are not supported.

23.5 Nest the Descriptive Flexfield Application Module Instance in the Application Module

You must nest the descriptive flexfield application module instance under the product application module before you can incorporate the descriptive flexfield usage into the UI.

You need to nest only one flexfield application module for a flexfield usage, even if two or more view links exist for the same flexfield usage.

23.5.1 How to Nest the Descriptive Flexfield Application Module Instance in the Application Module

You use the overview editor for your application module to nest the descriptive flexfield application module instance. The descriptive flexfield application module instance that you nest in the product application module shares the same transaction and entity object caches as the application module.

Before you begin:

1. You should have already created the product application module. For information about creating application modules, see the Implementing Business Services with Application Modules chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.
2. Create the master view object, which contains only nonflexfield attributes.
3. Create a flexfield business component for the descriptive flexfield usage as described in How to Create Descriptive Flexfield Business Components.

To nest the descriptive flexfield application module instance in the product application module:

1. In the Application Navigator, double-click the product application module.
2. Click the Data Model navigation tab.
3. On the Data Model Components page, expand the Application Module Instances section, as shown in Figure 23-14.

Figure 23-14 Application Module — Application Module Instances Section

4. In the Available tree, find and expand the applicationModule instance under the flexfield usage’s package. This is the package that you specified when you defined the entity details, as described in How to Register Entity Details.
5. Select the application module for the descriptive flexfield and move it to the Selected tree.

This application module was created when you created the flexfield business component and was named using the prefix that you specified when you defined the usage’s entity details, as described in How to Register Entity Details. For example, if you registered the `CasesDFF` prefix, the application module name is `CasesDFFAM`.

The New App Module Instance field under the list shows the name that will be used to identify instance. You can change this name.

### 23.6 Add a Descriptive Flexfield View Object to the Application Module

You must add a flexfield view object instance that reflects the hierarchy of the view link that you created in How to Create Descriptive Flexfield View Links to the application module for your application.

#### 23.6.1 How to Add a Descriptive Flexfield View Object Instance to the Application Module

Edit the product application module to add the flexfield view object.

**Before you begin:**

1. You should have already created the product application module. For information about creating application modules, see the Implementing Business Services with Application Modules chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

2. Create the master view object, which contains only nonflexfield attributes.

3. Create the flexfield business components for the descriptive flexfield usage as described in How to Create Descriptive Flexfield Business Components.

4. Create the flexfield view link as described in How to Create Descriptive Flexfield View Links.

**To add a descriptive flexfield view object to the product application module:**

1. In the Application Navigator, double-click the product application module.

2. Click the Data Model navigation tab.

3. In the View Object Instances section, select the master view object from the Available View Objects tree and click the right arrow to add it to the Data Model tree. Next, select the child view object for the flexfield view link and click the right arrow to add it to the Data Model tree, as shown in Figure 23-15.
23.7 Add Descriptive Flexfield UI Components to a Page

To include a descriptive flexfield on an application page, add the flexfield UI component to the page, and then configure the properties of the UI component.

**Note:**

You can also use descriptive flexfields in the following ways:

- Incorporate descriptive flexfield view object attributes as search criteria in an advanced query search form.
  
  See Incorporating Descriptive Flexfield into a Search Form.

- Use ADF Desktop Integration to incorporate descriptive flexfields into an Excel workbook.
  
  See Accessing Descriptive Flexfields from an ADF Desktop Integration Excel Workbook.

To add a descriptive flexfield UI component, you add the component to a page in the one of the following configurations:

- As part of a form component.
- As part of a table component, with the context-sensitive segments of the flexfield presented in a detailStamp facet. This is the typical configuration, which allows for the full range of possible values in the context segment.
- As part of a table component, with context-sensitive segments of the flexfield presented as columns. To use this configuration, you must guarantee that the flexfield context segment will have the same value in all rows of the table.

**Note:**

You cannot use a descriptive flexfield in a tree table component.
If your ADF Table is in an Applications Table component, you must add the following functionality to the UI:

- Create Row and Delete Row functionality if you are using your own **CreateInsert** button to create new rows.
- Empty table handling if you are using a custom **createInsert** method.
- Dynamic refresh of the context-sensitive segment columns whenever the Applications Table component is refreshed by another component, such as a button or a search query.

**Note:**
The following procedures assume that you are using the data-first approach of adding flexfields to your application. The UI-first approach is also available, but is not documented here. For more information, see the **Introduction to Placeholder Data Controls** section in the Developing Fusion Web Applications with Oracle Application Development Framework.

### 23.7.1 How to Add a Descriptive Flexfield UI Component to a Form

Use this procedure to incorporate your descriptive flexfield into a basic form.

**Before you begin:**

1. Create the descriptive flexfield view object as described in How to Create Descriptive Flexfield Business Components.
2. Create the view link as described in How to Create Descriptive Flexfield View Links.
3. Nest the descriptive flexfield application module instance in the product application module as described in How to Nest the Descriptive Flexfield Application Module Instance in the Application Module.
4. Add the view object instance to the application module as described in How to Add a Descriptive Flexfield View Object Instance to the Application Module.

**To add a descriptive flexfield UI component to a form:**

1. From the Data Controls panel, select the master view object and drag it onto the page to create the UI for the master view object.
2. When prompted, select **ADF Form** or select **Applications > Panel**.
3. In the Data Controls panel, expand the master view object and find the flexfield view object, as shown in Figure 23-16.

**Caution:**
You must use the flexfield view object that is the child of the master view object. Do not use the flexfield view object from the flexfield’s application module data control.
4. Drag the flexfield view object onto a form, and select the appropriate flexfield UI component.

**Tips:**

- If you place the flexfield in its own tab, header, or subheader, and you cannot provide a specific label for the region, consider using the label "Additional Information," which is the standard generic label in such a case for Oracle Fusion Applications.

- If a descriptive flexfield is in a region, such as a header, subheader, or tab, that does not contain nonflexfield fields, there is a possibility that the implementor will not use the flexfield segments and the region will be empty. To avoid the display of an empty region on the page, add controlling logic to hide the region if the implementor has not defined the segments that appear in the region. For information about how to determine if a segment has been defined, see How to Determine Whether Descriptive Flexfield Segments Have Been Defined. For information about using an EL expression to hide the region, see the Creating EL Expressions section in the Developing Web User Interfaces with Oracle ADF Faces.

- You can place the segments in a multiple-column layout. Use a multiple-column layout when the number of segments that will be added by the implementor is unknown and you anticipate that a large number of segments will be used. Otherwise, a flexfield with several segments will take up a large amount of space and the end user will have to scroll to see any fields that appear below the flexfield.
5. Optionally, to add Create Row and Delete Row functionality to the UI, drag the appropriate operation of the master view object from the Data Controls panel onto the page.

    **Tip:**
    You can configure the descriptive flexfield UI component to present multiple descriptive flexfield rows using the Panel Form Layout component, with each row's content based on a different context value. For more information, see the *Arranging Content in Forms* section of the *Developing Web User Interfaces with Oracle ADF Faces*.

### 23.7.2 How to Add an Unrestricted Descriptive Flexfield UI Component to a Table

Use this procedure to allow for the full range of possible context segment values. End users will be able to expose or hide the context-sensitive segments of the flexfield separately for each row of data.

**Before you begin:**

1. Create the descriptive flexfield view object as described in *How to Create Descriptive Flexfield Business Components*.

2. Create the view link as described in *How to Create Descriptive Flexfield View Links*.

3. Nest the descriptive flexfield application module in the product application module as described in *How to Nest the Descriptive Flexfield Application Module Instance in the Application Module*.

4. Add the view object instance to the application module as described in *How to Add a Descriptive Flexfield View Object Instance to the Application Module*.

**To add an unrestricted descriptive flexfield UI component to a table:**

1. From the Data Controls panel, select the master view object and drag it onto the page to create the UI for the master view object.

2. When prompted, select *ADF Table* or *Applications > Table*. You must select the **Row Selection** option, and set the appropriate width. Follow the wizard to create tree structure, **pathStamp**, and **nodeStamp** components.

3. In the Data Controls panel, expand the master view object and find the flexfield view object.

    **Caution:**
    You must use the flexfield view object that is the child of the master view object. Do not use the flexfield view object from the flexfield's application module data control.

4. Drag the flexfield view object onto the table on the design tab, as shown in *Figure 23-17*, and select *Oracle Descriptive Flexfield Column* as the UI component. This creates the base flexfield column in the table that the global segments will render.
5. Create a detail region (detailStamp facet) if the table does not have one, as shown in Figure 23-18.

6. Add a panel layout control to the detailStamp facet if you do not already have one.

7. Drag and drop the same flexfield view object into the detail region on the Source tab or the structure view as shown in Figure 23-19; this creates fields for the context-sensitive segments.
23.7.3 How to Add Descriptive Flexfield Context-Sensitive Segments to a Table as Columns

Typically, context-sensitive segments in a table are visible only in a detailStamp facet. This is because the flexfield context segment can contain a different value in each table row; therefore the set of associated context-sensitive segments that appears can vary from row to row. There is no way to present all of these varying results in a predefined column format within a single table.

However, if you can guarantee that a given context segment in every row of the table will always contain the same value for a given application page, the resulting combination of corresponding context-sensitive segments in each row will remain constant. In this circumstance the context-sensitive segments can be displayed as table columns.

For example, if the context segment is a country code, and the purpose of this application page is to manage only Italian tax data, then the context value should always be IT. The table columns for the context-sensitive segments will always be displayed in the configuration appropriate to the Italian tax system.

Before you begin:
1. Create the descriptive flexfield view object as described in How to Create Descriptive Flexfield Business Components.

2. Create the view link as described in How to Create Descriptive Flexfield View Links.

3. Nest the descriptive flexfield application module in the product application module as described in How to Nest the Descriptive Flexfield Application Module Instance in the Application Module.

4. Add the view object instance to the application module as described in How to Add a Descriptive Flexfield View Object Instance to the Application Module.

To add a descriptive flexfield UI component to a table as a column:

1. From the Data Controls panel, select the master view object and drag it onto the page to create the UI for the master view object.

2. When prompted, select ADF Table or Applications > Table. You must select the Row Selection option, and set the appropriate width.

3. In the Data Controls panel, expand the master view object and find the flexfield view object.

   Caution:
   You must use the flexfield view object that is the child of the master view object. Do not use the flexfield view object from the flexfield’s application module data control.

4. Drag the flexfield view object onto the table on the design tab as shown in Figure 23-17, and select Oracle Descriptive Flexfield Column as the UI component. This creates the base flexfield column in the table.

   Caution:
   Do not drop the flexfield view object into an existing column. The displaying of descriptive flexfields in the cell of a table column is not supported.

5. Edit the JSP source code for this page, and remove the following attribute from the <fnd:descriptiveFlexfield> tag:

    mode="global"

   See Table 23-2 for information about the mode attribute.

6. Add a WHERE clause or view criteria to the master view object to enforce the same predetermined context value for all rows of the table.

   If the context segment will be visible on the page, you must configure the segment to be read-only so end users cannot modify it. For more information, see How to Configure Segment-Level UI Properties.
23.7.4 How to Add Create Row and Delete Row Functionality to the Page

If your ADF Table is wrapped in an Applications Table component, and if you are using your own Create Insert button to create new rows, you must complete the following steps.

You do not need to complete these steps if new rows are created using the Applications Table’s New button or the New option on the Actions menu.

Caution:
If you enable end users to add new flexfield rows to the UI table, you can permit them to enter their own unique key values for a new row. However, you must provide a programmatically generated primary key value for the new row, otherwise it will generate an error.

To add Create Row and Delete Row functionality to the page:

1. From the Data Controls panel, drag the appropriate operation of the master view object (such as CreateInsert) onto the page.

2. Delete the newly created button, but not its pageDef entry. Example 23-2 shows an example of the pageDef entry.

3. Add a new button to the layout.

4. In the Property Inspector for the button, expand the Common section and set the ActionListener to the EL expression for the method binding. For example, #{myBean.customCreateInsert}.

Example 23-2 shows how the executables element of the page definition might look. Dff1RefInstanceIterator is the iterator of the master view object.

5. To ensure that new descriptive flexfield rows appear in the UI, add code to the application page (in the Invoke Application phase or just before the Render Response phase) to set the state of each newly created row to STATUS_NEW, as demonstrated in Example 23-3.

6. Add code to the custom createInsert method to handle an empty table as described in How to Add a Row to an Empty Table in a Custom createInsert Method

Example 23-2  Executables Element of Page Definition Code

```xml
<executables>
  <iterator Binds="Dff1RefInstance" RangeSize="25" DataControl="Dff1RefAMDataControl" id="Dff1RefInstanceIterator"/>
  <iterator Binds="Dff1Instance" RangeSize="25" DataControl="Dff1RefAMDataControl" id="Dff1InstanceIterator"/>
</executables>
```

Example 23-3  Code to Set New Table Row State to STATUS_NEW

```java
public void invokeMethod(String expr, Class[] paramTypes, Object[] params) {
    FacesContext fc = FacesContext.getCurrentInstance();
```
MethodBinding mb = fc.getApplication().createMethodBinding(expr,paramTypes);
return mb.invoke(fc,params);
}

public void customCreateInsert(ActionEvent actionEvent)
{
    invokeMethod("#{bindings.CreateInsert.execute}",ActionEvent.class,actionEvent);

    FacesContext fCtx = FacesContext.getCurrentInstance();
    javax.faces.application.Application app = fCtx.getApplication();
    ELContext elCtx = fCtx.getELContext();
    int index = 0;
    Row coreVORow =
        ((RowSetIterator)app.evaluateExpressionGet(fCtx,
        "#{bindings.Dff1RefInstanceIterator.rowSetIterator}",
        RowSetIterator.class)).getRowAtRangeIndex(index);
    coreVORow.setNewRowState(Row.STATUS_NEW);
}

Caution:

• If you enable end users to add new flexfield rows to the UI table, you
  must ensure that the default value of the new row's context segment is
  your predetermined value, matching the existing rows.

• You can permit end users to enter their own unique key values for a new
  row. However, you must provide a programmatically generated primary
  key value for the new row, otherwise it will generate an error.

• The context segment value in any existing row must not change at
  runtime. You must enforce this by hiding the context segment or by
  configuring it as read-only. For more information, see How to Configure
  Flexfield-Level UI Properties and How to Configure Segment-Level UI
  Properties.

23.7.5 How to Add a Row to an Empty Table in a Custom createInsert Method

If you are using a custom createInsert method to add rows to an Applications
Table component, you must include code similar to Example 23-4 to handle an empty
Table. Replace FirstRowInTable with logic to determine whether the table is empty and
the new row is the first row in the table.

Example 23-4    Handling Empty Tables in a Custom createInsert Method

if (FirstRowInTable &&
BindingUtil.getCustomProperty(table, "flexenabled") != null)
{
    List<UIComponent> columns = table.getChildren();
    if (columns != null)
    {
        for (UIComponent column: columns)
        {
            if (column.getChildCount() > 0)
            {
                UIComponent c1 = column.getChildren().get(0);
                if (c1 instanceof DescriptiveFlexfield)
23.7.6 How to Dynamically Refresh a Descriptive Flexfield

If your flexfield is in an ADF Table that is wrapped in an Applications Table component that is refreshed by another component, such as a button or a query, then you must add functionality to dynamically refresh the flexfield segments.

To refresh the flexfield segments based on the current iterator rowset data, create a listener method in the flexfield’s backing bean and bind the listener to the component that is initiating the table refresh. The listener must first call the default listener and then call `DescriptiveFlexfield.updateFlexColumns(RichTable)`, where `RichTable` is the binding for the table that contains the flexfield.

Example 23-5 shows an example of a custom flexfield handler for a query event. Example 23-6 shows the binding of the custom flexfield handler to the query component.

**Note:**

You do not need to handle flexfield refresh for standard Applications Table create and delete operations. However, custom create and delete operations must handle the refreshing of flexfields.

---

**Example 23-5  Flexfield Listener**

```java
public void customDffSearchQueryListener(QueryEvent queryEvent)
{
    FacesContext context = FacesContext.getCurrentInstance();
    String queryListener = "#{bindings.Dff3RefVOCriteriaQuery.processQuery}";
    ELContext elctx = context.getELContext();
    ExpressionFactory elFactory = context.getApplication().getExpressionFactory();
    MethodExpression queryMethod =
        elFactory.createMethodExpression(elctx, queryListener, Object.class,
                                         new java.lang.Class[] { QueryEvent.class });
    queryMethod.invoke(elctx, new Object[] { queryEvent });
    // appTable is the handle of the binding for the table
    // that contains the flexfield
    DescriptiveFlexfield.updateFlexColumns(appTable);
}
```

**Example 23-6  Binding the Flexfield Listener to the Search Query**

```xml
<af:query id="qryId1"
    headerText="#{applcoreBundle.QUERY_SEARCH_HEADER_TEXT}"
    disclosed="true"
    value="#{bindings.criteriaQuery.queryDescriptor}"
    model="#{bindings.criteriaQuery.queryModel}"
    queryListener="#{backingBeanScope.dffBean.customDffSearchQueryListener}"
```
23.7.7 What Happens When You Add a Descriptive Flexfield to a Page

Descriptive flexfield segments appear on a form as a widget with a implementor-defined label, just as core fields do. In a table, the label of the flexfield segment is the column header and the values are within each cell of the column.

Figure 23-20 shows an example of a descriptive flexfield used in a form on an application page.

Figure 23-20  Example of a Descriptive Flexfield in a Form

Figure 23-21 shows an example of a descriptive flexfield used in a table on an application page:

Figure 23-21  Example of a Descriptive Flexfield in a Table

Note:

Descriptive flexfield segments always appear as form fields or table columns in the same order that their corresponding attributes appear in the underlying view object.

23.8 Configure Descriptive Flexfield UI Components

You can control your descriptive flexfield’s behavior in the application UI by modifying properties at the flexfield level and the segment level, configuring descriptive flexfield parameters, and configuring the flexfield to handle value change events.
23.8.1 How to Configure Flexfield-Level UI Properties

You configure flexfield-level behavior by configuring the UI component’s properties.

Before you begin:
Add the descriptive flexfield to the page as described in Adding Descriptive Flexfield UI Components to a Page.

To configure flexfield-level properties:

- Select the descriptive flexfield’s UI component on the page and modify its properties in the Property Inspector, as shown in Figure 23-22.

Figure 23-22  Descriptive Flexfield Property Inspector

The significant properties on the Common, Data, Style and Behavior property tabs are listed in Table 23-2.

Table 23-2  Descriptive Flexfield Properties

<table>
<thead>
<tr>
<th>Tab &gt; Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common &gt; Id</td>
<td>This is the ID of the flexfield.</td>
</tr>
<tr>
<td>Common &gt; Rendered</td>
<td>This indicates whether the flexfield is rendered on the page. Values can be True (default) or False. EL expressions are allowed.</td>
</tr>
<tr>
<td>Common &gt; Value</td>
<td>This is the value of the flexfield. This should be an EL expression pointing to an iterator object. The iterator value must be statically declared in the page definition. This field is also visible on the Data tab.</td>
</tr>
<tr>
<td>Data &gt; Accessor</td>
<td>This property is the name of the accessor between the product view object and the flexfield view object.</td>
</tr>
<tr>
<td>Tab &gt; Property</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Data &gt; Category</td>
<td>This defines which category will be rendered on the page. The category can be set on each attribute’s custom property.</td>
</tr>
<tr>
<td>Style &gt; StyleClass</td>
<td>This property is the style class of the flexfield. A style class allows you to group a set of inline styles.</td>
</tr>
<tr>
<td>Style &gt; InlineStyle</td>
<td>This is the inline style of the component. The InlineStyle property is a string of CSS styles that can set individual properties such as background color, font style, or padding.</td>
</tr>
<tr>
<td>Behavior &gt; Read-Only</td>
<td>This indicates whether the flexfield is rendered as read-only. Values can be True or False (default). EL expressions are allowed.</td>
</tr>
<tr>
<td>Behavior &gt; Mode</td>
<td>This defines the UI mode of the descriptive flexfield component, to render all of the segments or just some of them. Values can be:</td>
</tr>
<tr>
<td></td>
<td>• No value (default): render all of the descriptive flexfield segments.</td>
</tr>
<tr>
<td></td>
<td>• global: render only the global segments of the flexfield. This is the default value for a descriptive flexfield inside a table column, to generate subcolumns for the global segments.</td>
</tr>
<tr>
<td></td>
<td>• contextSensitive: render only the context-sensitive segments of the flexfield (including the context segment). This is the default value for a descriptive flexfield inside a table detail region, to render the context values.</td>
</tr>
<tr>
<td>Behavior &gt; partialTriggers</td>
<td>This property is the IDs of the components that should trigger a partial update in the flexfield (String[]). EL expressions are allowed.</td>
</tr>
<tr>
<td>Behavior &gt; valueChangeListener</td>
<td>This property is a method reference to a value change listener (javax.faces.el.MethodBinding). It requires an EL expression.</td>
</tr>
<tr>
<td>Behavior &gt; binding</td>
<td>This is an EL reference that will store the component instance on a bean (oracle.apps.fnd.appcore.flex.ui.DescriptiveFlexfield). This property requires an EL expression.</td>
</tr>
</tbody>
</table>
Table 23-2 (Cont.) Descriptive Flexfield Properties

<table>
<thead>
<tr>
<th>Tab &gt; Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other &gt; showRequired</td>
<td>This indicates whether to bypass client-side validation of required segments that have null values. Values can be True or False (default). If the value is True, the end user can submit a form with null values in the fields for required segments (the segment fields that have asterisks). EL expressions are allowed. For related information about server-side validation of required segments, see How to Prevent Validation of Required Segments.</td>
</tr>
</tbody>
</table>

1 For a descriptive flexfield that was added as table columns, you cannot control this property on a row-by-row basis. It must be set to apply to the entire column.

23.8.2 How to Configure Segment-Level UI Properties

Descriptive flexfields support finer control of segments in the UI through the following segment-level UI properties:

- **Rendered**: This boolean property indicates whether the segment is visible on the application page. You can set this property with a literal value or an EL expression.

- **Required**: This boolean property indicates whether the segment must have a value. You can set this property with a literal value or an EL expression.

- **ReadOnly**: This boolean property indicates whether end users can modify the segment. You can set this property with a literal value or an EL expression.

- **Visible**: This boolean property specifies whether the segment is displayed in the page or is hidden. You can set this property with a literal value or an EL expression.

- **AutoSubmit**: This boolean property specifies whether a component automatically submits when a segment’s value changes. You can set this property with a literal value or an EL expression.

- **Label**: This string property provides a display label for a segment.

- **ShortDesc**: This string property provides a short description of a segment.

- **Columns**: This integer property specifies the width of the control in terms of the default font size of the browser. You can set this property with a literal value or an EL expression.

The default values of these properties are derived from the flexfield metadata, but you can override them by inserting the following flexfield hint components from the Applications page of the Component Palette:

- **Flexfield Context Segment Hints**: Use this component to configure the context segment.
• **Flexfield Segment Hints**: Use this component to configure all global segments or to configure all context-sensitive segments. Also use this component as a container for **Flexfield Segment Hint**.

• **Flexfield Segment Hint**: Nest this component in a **Flexfield Segment Hints** component to configure an individual global or context-sensitive segment.

---

**Note:**

If you set a segment’s **Required** property to **True**, for validation purposes you cannot override this by resetting it to **False** in the flexfield hint component. You can, however, do the reverse: Change a nonrequired segment to required in the flexfield hint component.

---

For information about using EL expressions, see the Creating ADF Data Binding EL Expressions section in the Developing Fusion Web Applications with Oracle Application Development Framework.

---

**Before you begin:**

Add the descriptive flexfield to the page as described in Adding Descriptive Flexfield UI Components to a Page.

---

**23.8.2.1 Configure a Context Segment**

Use **Flexfield Context Segment Hints** to set the **Visible**, **ReadOnly**, **Rendered**, **Required**, **Label**, **ShortDesc**, **Columns**, and **AutoSubmit** UI properties for a context segment.

**To configure a context segment:**

1. From the Applications page of the Component Palette, drag **Flexfield Context Segment Hints** and drop it as a child of the **Descriptive Flexfield** component.
2. In the Property Inspector, set the context segment’s UI properties.

---

**23.8.2.2 Configure All Global Segments**

Use **Flexfield Context Hints** to set the **Visible**, **ReadOnly**, **Rendered**, and **Required** properties to the same values for all the global segments.

**To configure all global segments:**

1. From the Applications page of the Component Palette, drag **Flexfield Segment Hints** and drop it as a child of the **Descriptive Flexfield** component.
2. In the Property Inspector, set the **Visible**, **ReadOnly**, **Rendered**, and **Required** properties.
3. Ensure that the **ContextCode** property is blank.

---

**Caution:**

For a descriptive flexfield that was added as table columns, you cannot configure the **Rendered** property of global segments on a row-by-row basis. It must be set to apply to the entire column.
23.8.2.3 Configure Individual Global Segments

Use a Flexfield Segment Hint component nested in a Flexfield Segment Hints component to set the Visible, ReadOnly, Rendered, Required, Label, ShortDesc, Columns, and AutoSubmit UI properties for a global segment.

You typically set the ReadOnly property to True if a default value is assigned to that segment, and you do not want end users to choose a value other than the default. Do not set both Required and ReadOnly to True if the segment does not have a default value.

Before you begin:

From the Applications page of the Component Palette, drag Flexfield Segment Hints and drop it as a child of the Descriptive Flexfield component. Do not set any properties.

To configure a global segment:

1. From the Applications page of the Component Palette, drag Flexfield Segment Hint and drop it as a child of Flexfield Segment Hints.
2. In the Property Inspector, set the SegmentCode property.

   Tip:

   To determine the correct value for the SegmentCode property, examine the FND_ACFF_SegmentName attribute of the segment’s viewAttribute element in the descriptive flexfield view object.

3. Set the global segment’s UI properties.

23.8.2.4 Configure All Context-Sensitive Segments

Use Flexfield Context Hints to set the Visible, ReadOnly, Rendered, and Required UI properties to the same values for all the context-sensitive segments.

To configure all global segments:

1. From the Applications page of the Component Palette, drag Flexfield Segment Hints and drop it as a child of the Descriptive Flexfield component.
2. In the Property Inspector, set the ContextCode property.
3. Set the Visible, ReadOnly, Rendered, and Required properties.

23.8.2.5 Configure Individual Context-Sensitive Segments

Use a Flexfield Segment Hint component nested in a Flexfield Segment Hints component to set the Visible, ReadOnly, Rendered, Required, Label, ShortDesc, Columns, and AutoSubmit UI properties for a context-sensitive segment.

You typically set the ReadOnly property to True if a default value is assigned to that segment, and you do not want end users to choose a value other than the default. Do not set both Required and ReadOnly to true if the segment does not have a default value.

Before you begin:
From the Applications page of the Component Palette, drag a **Flexfield Segment Hints** and drop it as a child of the **Descriptive Flexfield** component. Set the **ContextCode** property.

**To configure a context-sensitive segment:**

1. From the Applications page of the Component Palette, drag **Flexfield Segment Hint** and drop it as a child of **Flexfield Segment Hints**.

2. In the Property Inspector, set the **SegmentCode** property.

**Tip:**

To determine the correct value for the **SegmentCode** property, examine the `FND_ACFF_SegmentName` attribute of the segment's `viewAttribute` element in the descriptive flexfield view object.

3. Set the context-sensitive segment's UI properties.

### 23.8.3 How to Configure Descriptive Flexfield Parameters

You must add all the parameters that you have registered for the flexfield to the `partialTriggers` list so that each parameter's associated UI component is refreshed when its attribute is changed.

**Before you begin:**

1. Define and register the necessary parameters, if any.

2. Add the descriptive flexfield to the page as described in Adding Descriptive Flexfield UI Components to a Page.

**To configure descriptive flexfield parameters:**

1. Identify each UI component that corresponds (through the flexfield view object) to an entity object attribute to which a flexfield parameter is mapped, and ensure that the ID attribute of the UI component is set.

   For example, say you mapped the `Customer` parameter to the entity object `Customer_Name` attribute, which in turn has a corresponding view object attribute called `Customer_Name` and is displayed on the page using an inputText field with the prompt "Customer Name". You would ensure that this UI component has an ID, say, `customerInputText`.

2. In the **Behavior** section of the Property Inspector for the descriptive flexfield UI component, add the UI component ID to the list in the **PartialTriggers** field.

   In the previously introduced example, you would add `customerInputText` to the **PartialTriggers** list. **Example 23-7** shows the source view of the **partialTriggers** attribute.

**Example 23-7  Adding the UI Component ID to the PartialTriggers List**

```xml
<fnd:descriptiveFlexfield value="#{bindings.Dff1Iterator}"
    partialTriggers=customerInputText>
```
23.9 Testing Flexfields

You can test specified flexfields from JDeveloper using the Run with Flexfields dialog or the Debug with Flexfields dialog. You can also test the flexfields by running the application from an Integrated WebLogic Server instance. For more information, see Testing Flexfields.

23.10 Loading Seed Data

Any implementation of flexfields in Oracle Fusion Applications typically requires application seed data, which is the essential data to enable flexfields to work properly in applications. Flexfield seed data can be uploaded and extracted using the Seed Data Loader.

After you complete the registration process described in How to Register and Define Descriptive Flexfields, your flexfield seed data consists of the information that you registered for your flexfield, such as the tables and columns reserved for your flexfield. For a customer flexfield, the seed data contains only this registration data.

If your flexfield is a developer flexfield, you also serve the role of the implementor. In addition to the registration data, your flexfield seed data might include contexts, segments, and value sets that you have defined for your flexfield.

Use the Manage Descriptive Flexfields and Manage Value Sets tasks or use the PL/SQL APIs to create contexts, segments, and value sets, and then use the seed data loader to extract your seed data. To protect flexfield seed data from customer edits, use the PL/SQL flexfield registration API to mark a descriptive flexfield as protected, as described in Using the Setup APIs To Register and Define Descriptive Flexfields. Use the Manage Value Sets task to set the flag to protect a value set.

Only the user with the SEED_DATA_FROM_APPLICATION_USER user name can modify protected flexfields and value sets from the setup tasks, such as the Manage Descriptive Flexfields and Manage Value Sets tasks.

For information about extracting and loading seed data, see Initializing Oracle Fusion Application Data Using the Seed Data Loader.

23.11 Work with Descriptive Flexfield UI Programmatically

When working with descriptive flexfields programmatically, you might need to know how to do the following tasks:

- Update a descriptive flexfield
- Determine whether flexfield segments have been defined
- Configure a descriptive flexfield to handle value change events
- Configure a flexfield to prevent the validation of required segments.

23.11.1 How to Update a Descriptive Flexfield Programmatically

When you update a flexfield programmatically, you must obtain the same flexfield view row that is used by the UI. You use the getFlexfieldVORowFromEvent method to get a handle to a flexfield view row from the ValueChangeEvent instance.

```java
public static Row getFlexfieldVORowFromEvent(ValueChangeEvent vce);
```
Update the context value on the flexfield, not the master view row. Otherwise, the structure will not change. Do not update the entity object directly. The flexfield’s structure logic is in the setter of the view row, so do not bypass it.

23.11.2 How to Determine Whether Descriptive Flexfield Segments Have Been Defined

Your application might require information about any global or context-sensitive segments that exist in a descriptive flexfield’s metadata before it invokes a UI that includes the flexfield.

There is a view attribute in the descriptive flexfield view object, _FLEX_NumOfSegments, that contains the combined total number of global segments and context-sensitive segments in the flexfield. Its value is in the java.lang.Integer data format. This value may vary depending on the context.

The value of this view attribute is the number of segments defined in the metadata. For a given descriptive flexfield view row, a value of 0 means that only the context segment is available. Whether a segment is displayed is not taken into consideration.

23.11.3 How to Configure a Descriptive Flexfield to Handle Value Change Events

You can configure your application to recognize and respond to changes in individual descriptive flexfield segment values.

You register a value change listener to capture any ValueChangeEvent that occurs. When an end user changes a segment value, the input components associated with the flexfield segments on the application page deliver a value change event, and the listener is called.

Before you begin:

Add the descriptive flexfield to the page as described in Adding Descriptive Flexfield UI Components to a Page.

To configure a descriptive flexfield to handle value change events

1. Create the listener handler as a Java method (usually on a backing bean). Example 23-8 is an example of a method that handles a ValueChangeEvent.

Example 23-8 Sample Listener Handler Method

```java
public void dffChangeListener(ValueChangeEvent valueChangeEvent) {
    System.out.println("***** In dffChangeListener()");
    System.out.println("getSource() = " + valueChangeEvent.getSource());
    System.out.println("getOldValue() = " + valueChangeEvent.getOldValue());
    System.out.println("getNewValue() = " + valueChangeEvent.getNewValue());
}
```

2. Specify the handler method as UI metadata on the flexfield’s value change listener property (described in Table 23-2). In the Property Inspector, click the Edit button for the value change listener property, and a wizard appears to help you select an existing event listener or create a new listener. Example 23-9 is an example of the metadata as an EL expression that identifies the dffChangeListener listener from the Java method in Example 23-8.

Example 23-9 Sample EL Expression Identifying dffChangeListener

```xml
<fnd:descriptiveFlexfield value="#{bindings.PJCDFF1Iterator}" valueChangeListener="#{bindings.dffChangeListener}">
</fnd:descriptiveFlexfield>
```
For more information about handling value change events, see the Using Input Components and Defining Forms chapter in Developing Web User Interfaces with Oracle ADF Faces.

23.11.4 How to Prevent Validation of Required Segments

Sometimes you might need to bypass server-side validation of a required segment. For example, you might need to enable end users to save a partially filled form for later submission. To bypass server-side validation of a required segment, override the `getFlexfieldViewUsageConfiguration` method in the class for the master view object to disable this type of validation for the flexfield's view usage and view link accessor. In the case where you need to also bypass client-side validation, additionally set the flexfield's `showRequired` property to `true` as described in Table 23-2. When the `showRequired` property is `true`, the page displays asterisks next to the required segments to indicate that values are required, but will not perform client-side validation of segments with null values.

The `getFlexfieldViewUsageConfiguration` method, which is called whenever a flexfield view object is created through either a view link usage or a view link accessor, must accept the name of the view object and must return a Map object. As shown in Example 23-10, the method must verify that the name of the view object matches the name of the flexfield's master view object and that the name of the view link accessor matches the value passed into the method. If the names match, return a Map that contains a mapping for the key `FlexfieldViewObjectImpl.CONF_BYPASS_VALIDATION_REQUIRED`. The mapping's value must be boolean. A value of `Boolean.TRUE` indicates a null value should not cause a validation failure for any of the flexfield's required segments.

Example 23-10  `getFlexfieldViewUsageConfiguration` Method

```java
@Override
protected Map<String, Object> getFlexfieldViewUsageConfiguration(String accessorName) {
    if ("Parts1".equals(getName()) &&
        "FndCrmPartsDFFVL".equals(accessorName)) {
        Map<String, Object> map = new HashMap<String, Object>(1);
        map.put(            FlexfieldViewObjectImpl.CONF_BYPASS_VALIDATION_REQUIRED, Boolean.TRUE);
        return map;
    }
    else
    {
        return super.getFlexfieldViewUsageConfiguration(accessorName);
    }
}
```

23.12 Incorporate Descriptive Flexfield into a Search Form

You can include descriptive flexfield view object attributes as search criteria in an advanced query search form. This form enables end users to define ad hoc criteria to search for data in the application's master view object and its linked descriptive
flexfield view object. End users can select which attributes of the descriptive flexfield view object to use as search criteria.

**Note:**

If the search form displays the results in a table, the table displays the result set only under the following conditions:

- The end user performs a search.
- The **Run Automatically** check box for the search form's default search has been selected.

If a result set has not been displayed in the table, the only flexfield columns that appear in the table are the columns for the global segments.

### 23.12.1 How to Incorporate Descriptive Flexfields Into a Search Form

To incorporate a descriptive flexfield into a search form, use the **Edit Query Criteria** tab to add view criteria to a view object instance and then drop the view criteria onto the page as an ADF Query Panel with Table component.

**To add a descriptive flexfield to a search form:**

1. Add view criteria to the product view object instance that uses attributes from the view-linked descriptive flexfield view object, as shown in Figure 23-23.

**Figure 23-23  Edit Query Criteria Tab of View Object View Criteria Definition**
a. Enter a name for the view criteria definition and select the view accessor attribute from the master view object. The attributes of the view-linked descriptive flexfield view object appear.

b. Select the context segment (the discriminator) and use it as the attribute in the view criteria definition.

c. Select Equals as the **Operator** and select Literal as the **Operand**.

---

**Note:**

To include segments that use ID-based table-validated value sets or translated value sets in a query panel, the query mode must be set to **Both** to allow both database query and in-memory filtering. The search may be slow if the search results in a large number of rows fetched.

---

2. Create a new JSPX page in the user interface project.

   **Tip:**

   Oracle JDeveloper names the user interface project **ViewController** by default.

3. In the Data Control panel, select the named view criteria that you created previously.

4. Drag and drop the view criteria onto the page as an ADF Query Panel with Table component, as shown in **Figure 23-24**.

   **Figure 23-24** Page with Descriptive Flexfield Search Form

5. If the descriptive flexfield appears on the page as a column in a table, perform the following substeps to enable the execution of a saved search when the page is loaded for the first time:

   a. From the Property Inspector for the table in which the flexfield is displayed, obtain the **TableId** value.

   b. In the Property Inspector for the Query component, set the value of the **ResultComponentId** property to the **TableId** value.

6. Run the new search form page and click the **Advanced** button. When you click **Add Fields**, only the attributes associated with the base descriptive flexfield view object (global segments) are available as additional criteria. To include the context-sensitive attributes for a context, select the **equal to** operator for the **Context Value** criteria item and select a context. The **Add Fields** list refreshes to include the context-sensitive attributes from the subtype view object for the selected context, as shown in **Figure 23-25**.
Note:

Figure 23-8 illustrates the attributes associated with the base descriptive flexfield view object (global segments) and the context-sensitive attributes.

Figure 23-25    Descriptive Flexfield Search Form UI

For more information about working with search forms, see the Creating ADF Databound Search Forms chapter of Developing Fusion Web Applications with Oracle Application Development Framework.

23.13 Prepare Descriptive Flexfield Business Components for Oracle Business Intelligence

Oracle Business Intelligence is a comprehensive collection of enterprise business intelligence functionality that provides the full range of business intelligence capabilities including interactive dashboards, full ad hoc, proactive intelligence and alerts, enterprise and financial reporting, real-time predictive intelligence, and more.

Descriptive flexfields are modeled using polymorphic view objects, which are not compatible with Oracle Business Intelligence. To enable a descriptive flexfield to be used by Oracle Business Intelligence, the corresponding polymorphic view object must be flattened into a usable static form. When you create business components for this descriptive flexfield, the business component modeler recognizes the business intelligence-enabled setting, and a view object that is flattened for business intelligence is generated alongside the standard descriptive flexfield polymorphic view object. You must also slightly modify the process of creating descriptive flexfield view links and application modules.

When the business intelligence-enabled and flattened descriptive flexfield is configured as part of an application, the implementor can select which individual flexfield segments to make available for use with Oracle Business Intelligence. Only the segments that are business intelligence-enabled are included in the flattened view object.

23.13.1 How to Enable a Descriptive Flexfield for Oracle Business Intelligence

To enable implementors to perform business intelligence queries on whatever segments they configure for a flexfield, you must business-intelligence enable the
flexfield in one of the following ways. A flattened view object is generated only if the descriptive flexfield is business-intelligence enabled.

- Edit the flexfield from the Flexfield Registration Metadata wizard as described in Using the Flexfield Registration Metadata Wizard To Register and Define Descriptive Flexfields, and select the Enable Business Intelligence checkbox.
- Set the BI_ENABLED flag at registration time using the `fnd_flex_df_setup_apis.create_flexfield(...)` procedure. For information about using this procedure, see What You May Need to Know About the Descriptive Flexfield Setup API.
- Set the BI_ENABLED flag using the `fnd_flex_df_setup_apis.update_flexfield(...)` procedure. For information about using this procedures, see What You May Need to Know About the Descriptive Flexfield Setup API.

You can optionally provide flattened fact names for the flexfield’s entity details, as described in How to Register Entity Details. This helps to automate the process for importing the descriptive flexfield into Oracle Business Intelligence. If you are using `fnd_flex_df_setup_apis`, you provide the flattened fact name by setting the `BI_FLATTENED_FACT_NAME` value.

When a descriptive flexfield is business intelligence-enabled, implementors use the Manage Descriptive Flexfields task to enable the descriptive flexfield’s segments for business intelligence. Only the segments that are business intelligence-enabled are included in the flattened view object. Administrators must then import the changes into the Oracle Business Intelligence repository to make them available for Oracle Business Intelligence.

### 23.13.2 How to Flatten the Descriptive Flexfield Model for a Business Intelligence-Enabled Descriptive Flexfield

When you create business components for a business intelligence-enabled descriptive flexfield, the business component modeler recognizes the business intelligence-enabled setting. A view object that is flattened for Oracle Business Intelligence is generated alongside the standard descriptive flexfield polymorphic view object. You must also slightly modify the process of creating descriptive flexfield view links and application modules.

**Before you begin:**

1. Enable the flexfield and the desired segments for Oracle Business Intelligence as described in How to Enable a Descriptive Flexfield for.

2. If you have defined trees on any of the value sets that are referenced by the flexfield, ensure that the flattened tree view objects are already in your project. Otherwise, the Create Flexfield Business Components wizard that you use to create the flexfield business components will report the missing view objects as errors.

   For more information about flattened tree view objects, see About Designing a Column-Flattened View Object for Oracle Business Intelligence.

**To produce a business intelligence-enabled flattened descriptive flexfield model:**

1. Create descriptive flexfield business components as described in Creating Descriptive Flexfield Business Components.
For a flexfield that is business intelligence-enabled, the Create Flexfield Business Components wizard generates a business intelligence-specific view object and other business components under a directory called `analytics` in the package root directory. These are generated in addition to the standard descriptive flexfield view object.

2. Create a view link using the procedure described in `Creating Descriptive Flexfield View Links`. Keep the following in mind:

   • The master view object that you create with the standard wizard can be the same master view object that you create for the core descriptive flexfield model.

   • Create the view link from the master view object to the business intelligence-enabled flexfield base view object. The business intelligence-enabled flexfield is distinguished from the core flexfield by the prefix "BI:" as shown in Figure 23-26.

![Figure 23-26 Create Flexfield View Link Wizard — View Objects Page](image)

3. Create an application module for use with Oracle Business Intelligence as described in `Adding a Descriptive Flexfield View Object to the Application Module`. Make the following changes:

   a. On the Data Model page of the Create Application Module wizard, when you create an instance of the master view object, there is no need for a child view object.

   b. On the Application Modules page of the wizard, add an instance of the descriptive flexfield Oracle Business Intelligence application module as a nested instance of this application module. You can identify the Oracle Business Intelligence application module by the `analytics` subpackage under the package root.
4. Define the custom properties required to link the master view object instance to the default view instance inside the nested flexfield Oracle Business Intelligence application module instance.

This is done on the **General** tab of the nested business intelligence-enabled flexfield application module instance definition, as shown in Figure 23-27.

**Figure 23-27 Custom Properties for Business Intelligence-Enabled Application Module**

As you define the custom properties, keep the following points in mind:

- The default view instance inside the business intelligence-enabled flexfield application module is typically called **DefaultFlexViewUsage**.
- The custom property names should be formatted as **BI_VIEW_LINK_mypropertyname**
- The custom property values must be formatted as **source_viewobjectinstance_name, viewlink_definition_name, destination_viewobjectinstance_name**.
- Use the fully qualified view object instance names for the source view object and destination view object, and the fully qualified package name for the view link definition.
- Business intelligence joins between the view object instances that you specify in different application modules are created while importing them from Oracle ADF if custom properties are defined on the application module.

### 23.14 Publish Descriptive Flexfields as Web Services

You can make access to a descriptive flexfield available through web services, which will enable you to create, retrieve, update and delete operations on the flexfield data rows. You accomplish this by exposing utility methods in the product application module to access flexfield service data objects.
Utility methods for flexfield service data objects are helpful to the customers because the identifiers that are used in flexfield services are generally not the same as those in the flexfield definition, such as the customer-defined segment codes and context codes. For example, the name of the service data object for the customer-defined context "Product Type" might be called "CasesDFFProduct_Type." These utility methods help customers quickly identify a service data object or a property of the object by the identifiers with which the customers are familiar.

When you generate a flexfield business component, the descriptive flexfield business components and other artifacts are developed based on the information in the flexfield metadata. As illustrated in Figure 23-8, a base view object is created for the context and global segments. If any contexts have been configured, subtype view objects are generated for each configured context.

The example in Figure 23-28 shows a Business Component Browser view of a descriptive flexfield.

Figure 23-28  Business Component Browser View of a Descriptive Flexfield

To complete the development process to publish descriptive flexfields as web services:

1. Expose the descriptive flexfield as a web service.
2. Test the web service.

23.14.1 How to Expose a Descriptive Flexfield as a Web Service

To make available web service access to a descriptive flexfield, you must complete the following steps:

1. Service-enable the master view object.
2. Expose the application module as a web service.
3. Expose the operations on the master view object.
4. Add utility methods for the flexfield to the product application module. These utility methods, which are exposed to client applications, provide access to information from the FlexfieldSdoSupport object, which is not exposed to clients.

You can then deploy the service and run Java client programs to test the service as described in How to Test the Web Service.
For more information about service-enabling an application module, see *Integrating Service-Enabled Application Modules in Developing Fusion Web Applications with Oracle Application Development Framework*.

**Before you begin:**

1. Create a flexfield business component for the flexfield’s usage as described in *How to Create Descriptive Flexfield Business Components*.

   **Note:**
   
   When you generate a flexfield business component, the Create Descriptive Flexfield Business Components wizard automatically service-enables the business component by generating a Service Data Object (SDO) for the base view object and for every subtype view object.

2. Create a flexfield view link between the master view object, which contains only nonflexfield attributes, and the flexfield business component for the flexfield usage as described in *How to Create Descriptive Flexfield View Links*.

3. Nest the descriptive flexfield application module instance in the product application module as described in *How to Nest the Descriptive Flexfield Application Module Instance in the Application Module*.

4. Add the view object instance to the application module as described in *How to Add a Descriptive Flexfield View Object Instance to the Application Module*.

**To expose a descriptive flexfield as a web service:**

1. In the Application Navigator, open the view link between the master view object and the flexfield base view object.

2. In the overview editor, expand the Custom Properties section and add a SERVICE_PROCESS_CHILDREN property set to true, if one does not already exist.

3. Open the master view object, which is the view object that contains only the nonflexfield attributes.

4. In the overview editor click the *Java* navigation tab.

5. In the Java Classes section, click the *Edit* icon to generate and configure Java implementation classes.

6. In the Select Java Options dialog, select *Generate Service Data Object Class*, as shown in *Figure 23-29*.
7. Click OK.

8. From the Application Navigator, open the product application module.

9. In the overview editor, click the Web Service navigation tab.

10. If the Add icon is enabled, complete the following steps to service-enable the application module and expose the master view object operations:
   a. Click the Add icon to enable the application to support the service interface.
   b. In the Create Service Interface wizard, click Next three times to go to the Service View Instances page.
   c. Move the view instance for the master view object to the Selected list.
   d. Select the master view object from the Selected list as shown in Figure 23-30, select all the operations in the Basic Operations list, and click OK.
e. Click Finish.

11. If the Add icon is disabled, the application is already service enabled. Complete the following steps to expose the master view object operations:

a. In the Service Interface View Instances section, click the Edit icon.

b. Move the view instance for the master view object to the Selected list.

c. Select the master view object from the Selected list as shown in Figure 23-30, select all the operations in the Basic Operations list, and click OK.

12. Expand the Generated Files for Service Interface section, and make a note of the name of the remote server class for the product application module. This is the class that has a name ending with ServiceImpl.java.

13. In the overview editor for the product application module, click the Java navigation tab and click the link for the Application Module Class.

14. Add the utility methods for the flexfield service data objects to the product application module. Include methods that return the namespace and name of the service data object for a given context. Optionally, include methods that return the path of an attribute that is associated with a given segment in the service data object. Use self-explanatory names that reflect the name of the flexfield and what is returned. Example 23-11 shows example utility methods for the CasesDFF flexfield. Example 23-12 shows an example of the methods being invoked in a client application. The self-explanatory utility method names make it easy to understand the client code.
15. In the overview editor for the product application module, click the Web Service navigation tab and click the Edit icon in the Service Interface Custom Methods section.

16. In the Service Custom Methods dialog, move the newly added methods to the Selected list to make them available for clients and click OK.

The application module's remote server implementation class will be modified to expose these methods.

**Example 23-11  Utility Methods for Cases DFF Flexfield Service Data Object Support**

```java
// A private method to retrieve a FlexfieldSdoSupport object
// from the flexfield application module.
private FlexfieldSdoSupport getCasesDFFSdoSupport(String contextValue) {
    // Find the nested flexfield application module instance.
    DFFApplicationModuleImpl am = (DFFApplicationModuleImpl)findApplicationModule("CasesDFFAM1");
    // If contextValue is null, the base type is returned.
    return am.getSdoSupport(contextValue);
}

// Gets the namespace and name of the service data object for a
// given context value. Note how the method name and argument
// name are specific to CasesDFF.
public List<String> getCasesServiceNamespaceAndName(String productType) {
    FlexfieldSdoSupport ss = getCasesDFFSdoSupport(productType);
    if (ss == null) {
        throw new JboException("ERROR: Unable to fetch SDO for context " +
                               productType +
                               ".\nACTION NEEDED: Contact your system administrator. " +
                               "Ensure context " + productType +
                               " exists and the CasesDFF descriptive flexfield" +
                               " is deployed using Manage Descriptive Flexfields.");
    }
    return Arrays.asList(ss.getSdoNamespace(), ss.getSdoName());
}

// Gets the property path for "Product Type," which is the context
// segment of CasesDFF.
public String getCasesProductTypeServicePropertyPath() {
    FlexfieldSdoSupport ss = getCasesDFFSdoSupport(null);
    if (ss == null) {
        throw new JboException("ERROR: Unable to fetch SDO for context" +
                               ".\nACTION NEEDED: " +
                               "Contact your system administrator. " +
                               "Ensure that the CasesDFF descriptive " +
                               "flexfield is deployed using Manage Descriptive Flexfields.");
    }
    return ss.getDiscriminatorSdoPath();
}

// Gets the property paths for a list of segments specific to a
// product type.
public List<String> getCasesServicePropertyPaths(String productType,
                                                  List<String> segmentCodes) {
    FlexfieldSdoSupport ss = getCasesDFFSdoSupport(productType);
    return Arrays.asList();
}
```
if (ss == null) {
    throw new JboException("ERROR: Unable to fetch SDO for context " +
        productType +
        ".\nACTION NEEDED: Contact your system administrator. " +
        "Ensure context " + productType +
        " exists and the CasesDFF descriptive flexfield is " +
        "deployed using Manage Descriptive Flexfields.");
}

List<String> r = new ArrayList<String>(segmentCodes.size());
for (String segmentCode : segmentCodes) {
    r.add(ss.getSegmentSdoPath(segmentCode));
}
return r;

Example 23-12  Client Application for Updating CasesDFF in a CaseList Row

// Obtain the service definition for product type "Office Chair."
List<String> chairObjInfo=
    caseListService.getCasesServiceNamespaceAndName("Office Chair");

// Create a service data object for "Office Chair."
DataObject chairObj = dataFactory.create(chairObjInfo.get(0),
    chairObjInfo.get(1));

// Set the product type, so the product type matches the object type
chairObj.set(caseListService.getCasesProductTypeServicePropertyPath(),
    "Office Chair");

// Obtain the element names of the segments and set the values.
List<String> segmentPaths =
    caseListService.getCasesServicePropertyPaths("Office Chair",
    Arrays.asList("Broken Part"));

// Update the segment "Broken Part."
chairObj.set(segmentPaths.get(0), "Chair Back");

23.14.2 How to Test the Web Service

You can test the service by adding StringRefAddr elements to the Reference element for the product application module's service to the connections.xml file, deploying and manually testing the service, and optionally creating and running Java client programs to test the service.

Before you begin:

- Expose the descriptive flexfield as a web service as described in How to Expose a Descriptive Flexfield as a Web Service.
- Ensure that the BC4J Service Client and BC4J Service Runtime libraries are included in your project.

To test the web service:

1. Expand Application Resources, expand Descriptors, expand ADF Meta-INF, and open the connections.xml file.
2. Locate the Reference element for the product application module's service (DFF1MasterApplicationModuleService in this example).
3. Add the `StringRefAddr` elements that are shown in bold in Example 23-13 to the `Reference` element for the product application module's service. Modify the host and port number in the `jndiProviderURL` entry to point to Oracle WebLogic Server. The port number is typically 7101.

4. Run the remote server class for the product application module to deploy the service to Integrated WebLogic Server and to manually test the web service.

   **Note:**

   The remote server class was generated when you exposed the descriptive flexfield as a web service in *How to Expose a Descriptive Flexfield as a Web Service*. This class has a name that ends with `ServiceImpl.java`.

5. Optionally, create and run Java client programs to test invoking the web service:

   The following examples demonstrate how to write programs to test the web service.

   • Example 23-17 shows how to get a data row. The XML payload output of this program is shown in Example 23-18.

   • Example 23-19 shows how to create a new data row.

   • Example 23-20 shows how to update an existing data row.

   **Note:**

   You must provide the flexfield's `xsi:subtype` value in the payloads that are sent to the service. If you do not provide the `xsi:subtype` value, the attributes for that subtype are ignored, and will not be created or updated.

---

**Example 23-13  StringRefAddr Elements to Add to Service Reference in connections.xml**

```xml
<Reference name="{http://xmlns.oracle.com/oracle/apps/fnd/applcore/flex/test/dff1/model/}DFF1MasterApplicationModuleService" className="oracle.jbo.client.svc.Service" xmlns="" xmlns="">
   <Factory className="oracle.jbo.client.svc.ServiceFactory"/>
   <RefAddresses>
      <StringRefAddr addrType="serviceInterfaceName">
         <Contents>oracle.apps.fnd.applcore.flex.test.dff1.model.DFF1MasterApplicationModuleService</Contents>
      </StringRefAddr>
      <StringRefAddr addrType="serviceEndpointProvider">
         <Contents>ADFBC</Contents>
      </StringRefAddr>
      <StringRefAddr addrType="jndiName">
         <Contents>DFF1MasterApplicationModuleServiceBean#oracle.apps.fnd.applcore.flex.test.dff1.model.DFF1MasterApplicationModuleService</Contents>
      </StringRefAddr>
      <StringRefAddr addrType="serviceSchemaName">
         <Contents>DFF1MasterApplicationModuleService.xsd</Contents>
      </StringRefAddr>
      <StringRefAddr addrType="serviceSchemaLocation">
         <Contents>DFF1MasterApplicationModuleService.xsd</Contents>
      </StringRefAddr>
   </RefAddresses>
</Reference>
```
Example 23-14  Example Get Payload Input for Fusion Service Tester Utility

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/*">
  <soap:Body>
    <ns1:getCaseList1 xmlns:ns1="http://xmlns.oracle.com/apps/applcore/crmdemo/types/*">
      <ns1:id>100128</ns1:id>
    </ns1:getCaseList1>
  </soap:Body>
</soap:Envelope>

Example 23-15  Example Create Payload Input for Fusion Service Tester Utility

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/*">
  <soap:Body>
      <ns1:caseList1 xmlns:ns0="http://xmlns.oracle.com/apps/applcore/crmdemo/">
        <ns0:Id>99910</ns0:Id>
        <ns0:TypeName>Service Request</ns0:TypeName>
        <ns0:CustomerName>A Corporation</ns0:CustomerName>
        <ns0:StatusName>Priority</ns0:StatusName>
        <ns0:StatusIconClass>DemoStatusPriority</ns0:StatusIconClass>
        <ns0:StatusDate>2012-12-10</ns0:StatusDate>
        <ns0:DateInitiated>2011-12-28</ns0:DateInitiated>
        <ns0:LastModifiedDate>2012-12-10</ns0:LastModifiedDate>
        <ns0:ProblemDescription>Customer's K3000 Industrial Anvil from A Company is broken. They need help in fixing it.</ns0:ProblemDescription>
        <ns0:AssignedUserFirstName>Myfirst</ns0:AssignedUserFirstName>
        <ns0:AssignedUserLastName>Mylast</ns0:AssignedUserLastName>
        <ns0:AssignedUserMiddleInitial>M</ns0:AssignedUserMiddleInitial>
        <ns0:AssignedUserFullName>Myfirst M. Mylast</ns0:AssignedUserFullName>
        <ns0:CustomerId>10035</ns0:CustomerId>
        <ns0:TypeId>47</ns0:TypeId>
        <ns0:StatusIdIndex>2</ns0:StatusIdIndex>
        <ns0:AssignedUserIdIndex>10000</ns0:AssignedUserIdIndex>
        <ns0:StatusId>2</ns0:StatusId>
    </ns1:caseList1>
  </soap:Body>
</soap:Envelope>
Example 23-16  Example Update Payload Input for Fusion Service Tester Utility

<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:ns1="http://xmlns.oracle.com/apps/fnd/applcore/crmdemo/types/
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ns1:createCaseList1 xmlns:ns0="http://xmlns.oracle.com/apps/fnd/applcore/crmdemo/"
 xmlns:ns3="http://xmlns.oracle.com/apps/fnd/applcore/crmdemo/flex/cases/
 xsi:type="ns3:CasesDFFIndustrial__Anvil">
    <ns3:Id>99910</ns3:Id>
    <ns3:_CUSTOMER_EMAIL>myemail@example.com</ns3:_CUSTOMER_EMAIL>
    <ns3:_RESOLUTION_DATE xsi:nil="true"/>
    <ns3:_ESCALATION_LEVEL>3</ns3:_ESCALATION_LEVEL>
    <ns3:_FLEX_Context>Industrial Anvil</ns3:_FLEX_Context>
    <ns3:_FLEX_NumOfSegments>4</ns3:_FLEX_NumOfSegments>
    <ns3:_WEIGHT>60</ns3:_WEIGHT>
  </ns0:FndCrmCasesDFFVLA>
</soap:Body>
</soap:Envelope>
Example 23-17  Web Service Get Operation

```java
package oracle.apps.fnd.applcore.flex.test.dff1.model.test;

import ...

public class DFFTester {
private static final String PK1 = "MY_PRIMARY_KEY";
public DFFTester() {
    super();
}
public void getFKRowGivenKey() {
    try {
        DFF1MasterApplicationModuleService dff1Service =
            (DFF1MasterApplicationModuleService)
            ServiceFactory.getServiceProxy(DFF1MasterApplicationModuleService.NAME);
        List<String> dff1Info = dff1Service.getMyFlexfieldSdoNamespaceAndName(null);
        System.out.println(dff1Info);

        final String contextValue = "MY_CONTEXT_VALUE";
        List<String> dffInfo2 =
            dff1Service.getMyFlexfieldSdoNamespaceAndName(contextValue);
        System.out.println(dffInfo2);

        MasterDff masterVOSDO = dff1Service.getMasterDff1(PK1);
        DataObject dataObject = (DataObject) masterVOSDO;
        String uri = dataObject.getType().getURI();
        XMLHelper xmlhelper = ServiceFactory.getXMLHelper(dff1Service);
        String xml = xmlhelper.save(dataObject, uri, "MasterDff");
        System.out.println(xml);
    } catch (Exception e) {
        // Handle exception
    }
}
```

public static void main(String args[])
    {
        DFFTester dfftester = new DFFTester();
        dfftester.getFKRowGivenKey();
    }

Example 23-18  Example XML Payload Output of Web Service Get Operation

<?xml version="1.0" encoding="UTF-8"?>
    <ns2:CreatedBy>0</ns2:CreatedBy>
    <ns2:CreationDate>2009-12-03T23:39:25.0-08:00</ns2:CreationDate>
    <ns2:LastUpdateDate>2009-12-03T23:39:25.0-08:00</ns2:LastUpdateDate>
    <ns2:LastUpdateLogin>0</ns2:LastUpdateLogin>
    <ns2:LastUpdatedBy>0</ns2:LastUpdatedBy>
    <ns2:PrimaryKeyCode>VS_IND_COC_DEP_DAT_ON_DAT05</ns2:PrimaryKeyCode>
    <ns2:ProductContext xsi:nil="true"/>
    <ns2:ProductDate xsi:nil="true"/>
    <ns2:ProductNumber xsi:nil="true"/>
    <ns2:ProductVarchar2 xsi:nil="true"/>
    <ns2:.vs_5FIND_5FCOC_5FDEP_5FDAT_5FON_5FDAT><ns1:PrimaryKeyCode>VS_IND_COC_DEP_DAT_ON_DAT05</ns1:PrimaryKeyCode>
        <ns1:_FLEX_ValidationDate>2009-12-03</ns1:_FLEX_ValidationDate>
        <ns1:_GlobalSegment1>05</ns1:_GlobalSegment1>
        <ns1:_GlobalSegment2>Value05</ns1:_GlobalSegment2>
        <ns1:_GLOBAL_STATE_ID NUM xsi:nil="true"/>
        <ns1:_GLOBAL_STATE_ID NUM_Display xsi:nil="true"/>
        <ns1:_FLEX_Context>VS_IND_COC_DEP_DAT_ON_DAT</ns1:_FLEX_Context>
        <ns1:_FLEX_NumOfSegments>5</ns1:_FLEX_NumOfSegments>
        <ns1:_State>RI</ns1:_State>
        <ns1:_State_Holiday>2007-08-12</ns1:_State_Holiday>
    </ns1:vs_5FIND_5FCOC_5FDEP_5FDAT_5FON_5FDAT>
    <ns1:vs_5FIND_5FCOC_5FDEP_5FDAT_5FON_5FDAT>
</ns2:MasterDff>

Example 23-19  Web Service Create Operation

... public void createMasterDffRow() {
    try {
        DFF1MasterApplicationModuleService dff1Service =
            (DFF1MasterApplicationModuleService)
                ServiceFactory.getServiceProxy(
                    DFF1MasterApplicationModuleService.NAME);
        DataFactory dataFactory =
            ServiceFactory.getDataFactory(dff1Service);
        MasterDffImpl dffMaster =
            (MasterDffImpl)dataFactory.create(MasterDff.class);
        dffMaster.setPrimaryKeyCode(PK1);
        final String contextValue = "MY_CONTEXT";
        List<String> dffInfo =
            dff1Service.getMyFlexfieldSdoNamespaceAndName(contextValue);
        System.out.println(dffInfo);
        DataObject dffSubType =
            ...
Example 23-20 Web Service Update Operation

public void updateMasterDffRow() {
    try {
        DFF1MasterApplicationModuleService dff1Service = 
            (DFF1MasterApplicationModuleService)
            ServiceFactory.getServiceProxy(
                DFF1MasterApplicationModuleService.NAME);
        DataFactory dataFactory = 
            ServiceFactory.getDataFactory(dff1Service);

        MasterDffImpl dffMaster = 
            (MasterDffImpl)dff1Service.getMasterDff1(PK1);
        XMLHelper xmlhelper = ServiceFactory.getXMLHelper(dff1Service);
        String uri = dffMaster.getType().getURI();
        String xml = xmlhelper.save(dffMaster, uri, "MasterDff");
        System.out.println(xml);

        Object dffObject = dffMaster.getDff1();
        System.out.println("Dff Object Name->" +
            dffObject.getClass().getName());

        final String contextValue = "MY_CONTEXT";
        List<String> dffInfo = 
            dff1Service.getMyFlexfieldSdoNamespaceAndName(contextValue);
        System.out.println(dffInfo);
        DataObject dffSubType = 
            dataFactory.create(dffInfo.get(0), dffInfo.get(1));
        System.out.println(dff1Service.getMyFlexfieldTypeSdoPath());
        dffSubType.set(dff1Service.getMyFlexfieldSdoPath(), contextValue);
        dffMaster.setDff1(dffSubType);
        dff1Service.createMasterDff1(dffMaster);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
23.15 Accessing Descriptive Flexfields from an ADF Desktop Integration Excel Workbook

ADF Desktop Integration makes it possible to combine desktop productivity applications with Oracle Fusion Applications, so you can use a program like Microsoft Excel as an interface to access application data.

Using ADF Desktop Integration, you can incorporate descriptive flexfields into an integrated Excel workbook, so you can work with the flexfield data from within the workbook.

The Oracle Fusion Middleware Developing Applications with Oracle ADF Desktop Integration guide provides most of the information you need to complete the required activities, including the following:

- Preparing your development environment for desktop integration.
- Creating a page definition file that will expose the Oracle ADF bindings for use in Excel.
- Creating an Excel workbook to integrate with the descriptive flexfield.
- Preparing your Excel workbook to access the column containing the flexfield.
- Incorporating a descriptive flexfield as a dynamic component or a single cell on a worksheet in the workbook.

The Oracle Fusion Middleware Developing Applications with Oracle ADF Desktop Integration guide does not make explicit reference to flexfields. In addition to the standard implementation steps covered in that guide, you must modify your implementation to accommodate flexfields.

There are two ways to access a descriptive flexfield in Excel:

- Using a dynamic column in an ADF Desktop Integration Table.
A web page in a popup dialog can be associated with a dynamic column, enabling end users to pick flexfield segment values. Alternatively, users can enter values directly into the segment fields. No custom code is required in either case.

This is the most typical scenario. Each descriptive flexfield segment is displayed as a distinct column in the ADF Desktop Integration Table.

- Using a static column in a popup dialog associated with a single cell. Use this approach for either of the following reasons:
  - The descriptive flexfield is exposed in an ADF Desktop Integration Table, is context sensitive, and the context changes from row to row. A static column is required in this case.
  - You do not want descriptive flexfield segments to occupy too much space in the worksheet.

In addition to using the popup dialog, end users can enter values directly into the segment field, with the values separated by an appropriate delimiter that you specify.

**Note:**

A static column is any column for which the `DynamicColumn` property is set to `False`.

Individual flexfield segments do not appear in the worksheet. Instead, ADF Desktop Integration invokes a separate JSPX page on which the flexfield will be visible. You can use this scenario with an ADF Desktop Integration Form, or either table type.

The descriptive flexfield's segments are part of your database table, so the flexfield is generated against the same entity object on which your worksheet view object is based.

To complete the process for accessing descriptive flexfields from an ADF Desktop Integration Excel workbook:

1. Configure ADF Desktop Integration with either a dynamic or static column descriptive flexfield.
2. If using a dynamic column descriptive flexfield where the end user can control the context value, modify the application to update the descriptive flexfield structure whenever user-initiated context value changes occur in the dynamic column descriptive flexfield.
3. Create a custom method to process updates or inserts for descriptive flexfield data row and add code to invoke the method.

### 23.15.1 How to Configure ADF Desktop Integration with a Dynamic Column Descriptive Flexfield

When you configure the ADF Desktop Integration Table, make the following changes:

- Add the ADF Desktop Integration Model API library to your data model project.
• In your page definition for the worksheet, add the descriptive flexfield that you want to access to the master worksheet view object as a child node. Do not add any display attribute to the child node that expands as a dynamic column in the worksheet.

For more information about how to create a page definition file for an ADF Desktop Integration project, see the Working with Page Definition Files for an Integrated Excel Workbook section of the Developing Applications with Oracle ADF Desktop Integration.

• To make the descriptive flexfield column of the ADF Desktop IntegrationTable dynamic, set the DynamicColumn property in the TableColumn array of the ADF Desktop Integration Table to True. A dynamic column in the TableColumn array is a column that is bound to a tree binding or tree node binding whose attribute names are not known at design time. A dynamic column can expand to more than a single worksheet column at runtime.

For more information about the binding syntax for dynamic columns, see the Adding a Dynamic Column to Your ADF Table Component section of the Developing Applications with Oracle ADF Desktop Integration.

• For the table’s UpdateComponent and InsertComponent properties, specify one of the following as the subcomponent to use:
  - Inputtext
  - OutputText
  - ModelDrivenColumnComponent

• For the subcomponent’s Value property, access the Expression Builder, expand the Bindings node and your tree binding for the table, and select the flexfield node.

• For the subcomponent’s Label property, access the Expression Builder, expand the Bindings node and your tree binding for the table, and select the flexfield node.

23.15.2 How to Handle User-Initiated Context Value Changes in a Dynamic Column Descriptive Flexfield

ADF Desktop Integration requires that to use a dynamic column implementation, the structure of the descriptive flexfield must remain constant for all rows in a given result set. However, each time a new result set is downloaded into the table, the context value (and thus the structure) can be changed.

If the context value is set globally for the end user of the workbook, changes are not an issue. However, if the user can control the context value (for example, using a list of values (LOV) in a header form), your application must be able to respond appropriately to update the descriptive flexfield structure.

After the end user specifies a context value, you must invoke the worksheet UpSync method to get the new value into the model layer. Then you can use the ADF Desktop Integration Table Download method to get fresh data with the new descriptive flexfield structure.
Note:
For an insert-only table, the Download method is undesirable. For these cases, use either the ADF Desktop Integration Table DownloadForInsert method or the Initialize method to enable the ADF Desktop Integration Table to reconfigure to accommodate the new flexfield structure.

23.15.3 How to Configure ADF Desktop Integration with a Static Column Descriptive Flexfield

If the structure of your descriptive flexfield varies from row to row in a given result set, you cannot implement the flexfield as a dynamic column — it will produce errors. You must use a static column with a popup dialog.

Note:
If a specific dialog title cannot be provided because the configuration of the flexfield will not be known until implementation, use "Additional Information" for the title, which is the standard generic label in such a case for Oracle Fusion Applications.

ADF Desktop Integration supports descriptive flexfields by using tree bindings in an ADF Desktop Integration Table. If you desire, the table can be configured to be read-only.

Note:
A descriptive flexfield appears as a node in the tree binding at design time. Because flexfields are built up dynamically at runtime, you will not see any attributes under the flexfield node in the page definition, but the node itself is present.

When you configure the popup dialog, make the following changes:

- You can use the column’s action set properties to make the descriptive flexfield web page available for editing. Include the attributes used in the web page in the table’s cached attributes unless the row will be committed immediately.

- You must choose a fixed attribute (a descriptive flexfield global segment attribute) to represent the flexfield in the worksheet. Add a Dialog action to the DoubleClickActionSet component action of an InputText or OutputText component, then connect the Dialog action to JSPX page that will display the descriptive flexfield.

For more information about how to create a page definition file for an ADF Desktop Integration Table project, see the Working with Page Definition Files for an Integrated Excel Workbook section of the Developing Applications with Oracle ADF Desktop Integration.

For static display of a descriptive flexfield in an ADF Desktop Integration workbook, you must create an updatable transient attribute in the view object on which the ADF Desktop Integration Table is based. This transient attribute will hold the concatenated value of the descriptive flexfield segments, separated by a delimiter. If one purpose of
Accessing Descriptive Flexfields from an ADF Desktop Integration Excel Workbook

the worksheet is to display existing data from the database, the transient attribute
should be populated using custom application module methods upon returning from
a popup dialog or opening the worksheet.

23.15.4 How to Handle Updating or Inserting of a Descriptive Flexfield Data Row
To handle updating or inserting of a data row containing a descriptive flexfield in an
ADF Desktop Integration Table, you call a custom application module method that
contains appropriate code, as follows:
•

To update an existing row, add your code to the UpdateRowActionId property of
the table.

•

To insert a new row, add your code to the InsertAfterRowActionId property of
the table.

The context value should be set before calling the application module method, which
gets called in the doubleclickactionset component action of the table's
UpdateComponent or InsertComponent properties. This is applicable for both
dynamic column and static column display of descriptive flexfields. Setting the context
value appropriately is important because this controls the structure of the flexfield.
The following examples demonstrate the code needed to accomplish these tasks.
Example 23-21 and Example 23-22 apply to an ADF Desktop Integration
implementation with the descriptive flexfield exposed as a static column.
Example 23-23 presents the isSegmentDisplayable() method that is used in the
other two examples.
Example 23-21
Flexfield

Updating or Inserting a Row with a Static Column Descriptive

//Retrieve your worksheet view object
myworksheet_VOImpl srcVO = myVO
//Retrieve the current row from your view object.
//That is the row that is being processed by ADF Desktop Integration.
myworksheet_VO_Row_Impl srcRow = myworksheet_VO_Row_Impl srcVO.getCurrentRow();
//This gives the transient attribute value from your worksheet.
Object dffAttributeValue = srcRow.getAttribute(mytransientattribute);
//Get the descriptive flexfield row based on the descriptive flexfield view
accessor.
DFFViewRowImpl dffRow = (DFFViewRowImpl)srcRow.getAttribute(mydff_viewaccessor);
//Check if the single cell value is null.
if (dffAttributeValue != null && !("").equals(dffAttributeValue)) {
//Getting DFF metadata information
FlexfieldViewDefImpl dffImpl =
(FlexfieldViewDefImpl)dffRow.getFlexfieldViewDef();
String delim = dffImpl.getDelimiter();
//Parse the DFF string into tokens
StringTokenizer token =
new StringTokenizer(dffAttributeValue.toString(), delim);
while (token.hasMoreTokens()) {
prjValues.add(token.nextToken());
}
}

Use Descriptive Flexfields 23-73


//Get the descriptive flexfield segment information.
ListAttributeDef listSeg =
    dffRow.getFlexfieldViewDef().getFlexfieldAttributes();

Iterator listSegIterator = listSeg.iterator();
AttributeDef seg = null;
List<String> segDisplay = new ArrayList<String>();
while (listSegIterator.hasNext()) {
    seg = (AttributeDef)listSegIterator.next();
    if (isSegmentDisplayable(seg, dffRow)) {
        segDisplay.add(seg.getName());
    }
}

//Get the size of the segment.
int segValueSize = 0;
if (dffAttributeValue != null && !("").equals(dffAttributeValue))
    segValueSize = prjValues.size();
else
    segValueSize = segDisplay.size();

//This check is required to handle a context dependent DFF case.
//If the context is changed before uploading, do not proceed.
if (segValueSize < prjValues.size())
    return;
if (segDisplay.size() == 0)
    return;
for (int i = 0; i < segValueSize; i++) {
    if (dffAttributeValue != null && !("").equals(dffAttributeValue)) {
        if (prjValues.get(i) != null && !prjValues.get(i).equals("")) {
            dffRow.setAttribute(segDisplay.get(i), prjValues.get(i));
        } else {
            dffRow.setAttribute(segDisplay.get(i), null);
        }
    } else {
        dffRow.setAttribute(segDisplay.get(i), null);
    }
}

Example 23-22 Applying Modified Segment Values to a Cell in a Static Column
Descriptive Flexfield

Add this code as an application module method that will be invoked by clicking the OK button of a popup dialog. This method can also be used to populate transient attribute values used for single cell display upon opening the worksheet, if the worksheet is intended to display existing records from the database.

//Retrieve the DFF row through the DFF view accessor.
DFFViewRowImpl dffRow =
    (DFFViewRowImpl)row.getAttribute(  
        "DFF_viewaccessorattribute_from_worksheetVO");

//Get the delimiter information for your DFF.
FlexfieldViewDefImpl dffImpl =
    (FlexfieldViewDefImpl)dffRow.getFlexfieldViewDef();
String delim = dffImpl.getDelimiter();

//Get the segment information for your DFF.
ListAttributeDef listSeg =
    dffRow.getFlexfieldViewDef().getFlexfieldAttributes();
// Your DFF will have many segments, but not all of them will be used for display. 
// This code loops through DFF segments and obtains the name and type 
// of each displayable segment.
Iterator listSegIterator = listSeg.iterator();
AttributeDef seg = null;
List<String> segDisplay = new ArrayList<String>();
List<Integer> segDisplayType = new ArrayList<Integer>();

while (listSegIterator.hasNext()) {
    seg = (AttributeDef) listSegIterator.next();
    if (isSegmentDisplayable(seg, dffRow)) {
        segDisplay.add(seg.getName());
        segDisplayType.add(new Integer(seg.getSQLType()));
    }
}

int segDisplaySize = segDisplay.size();
StringBuffer segmentString = new StringBuffer();

// This loop is constructing a string out of displayed segment values
// with a delimiter in between each segment value.
// If the segment type is date, remove the time from the date.
// For the first segment (i=0), you need to handle it differently
// to construct the string correctly.
for (int i = 0; i < segDisplaySize; i++) {
    if (dffRow.getAttribute(segDisplay.get(i)) != null) {
        if (i == 0) {
            if (segDisplayType.get(i) == 91) {
                StringTokenizer stTime =
                        new StringTokenizer(dffRow.getAttribute(segDisplay.get(i)).toString(), " ");
                segmentString.append(stTime.nextToken());
            } else {
                segmentString.append(dffRow.getAttribute(segDisplay.get(i)));
            }
        } else {
            segmentString.append(delim);
            if (segDisplayType.get(i) == 91) {
                StringTokenizer stTime =
                        new StringTokenizer(dffRow.getAttribute(segDisplay.get(i)).toString(), " ");
                segmentString.append(stTime.nextToken());
            } else {
                segmentString.append(dffRow.getAttribute(segDisplay.get(i)));
            }
        }
    } else {
        if (i==0)
            segmentString.append(" ");
        else {
            segmentString.append(delim);
            segmentString.append(" ");
        }
    }
}
Example 23-23 isSegmentDisplayable() Helper Method Used in the Previous Examples

The input parameters for this method are the segment attribute definition and the descriptive flexfield row.

```java
public boolean isSegmentDisplayable(AttributeDef seg,
                                       DFFViewRowImpl dffRow) {
    if (seg.getProperty("FND_ACFF_DisplayAttributeName") == null) {
        if (seg.getProperty(seg.getUIHelper().ATTRIBUTE_DISPLAY_HINT) ==null ||
            !seg.getProperty(seg.getUIHelper().ATTRIBUTE_DISPLAY_HINT).equals("Hide")) {
            return true;
        } else {
            return false;
        }
    } else {
        int attrIndex =
            dffRow.getFlexfieldViewDef().getAttributeIndexOf(
                seg.getProperty("FND_ACFF_DisplayAttributeName").toString());
        AttributeDef displayAttrDef =
            dffRow.getFlexfieldViewDef().getAttributeDef(attrIndex);
        if (displayAttrDef.getProperty(
                seg.getUIHelper().ATTRIBUTE_DISPLAY_HINT) == null ||
            !displayAttrDef.getProperty(
                seg.getUIHelper().ATTRIBUTE_DISPLAY_HINT).equals("Hide")) {
            return true;
        } else {
            return false;
        }
    }
    return false;
}
```
Use Extensible Flexfields

This chapter discusses how to use extensible flexfields to enable customers to add additional attributes to application business objects in Oracle Fusion Applications. This chapter includes the following sections:

- Introduction to Extensible Flexfields
- Overview of Integrating Extensible Flexfields in an Application
- Creating Extensible Flexfield Data Tables
- Registering Extension Tables as Secured Objects
- Creating and Configuring Extensible Flexfield Entity Objects
- Creating and Configuring Extensible Flexfield View Objects
- Defining and Registering Extensible Flexfields
- Creating Extensible Flexfield Business Components
- Employing an Extensible Flexfield on a User Interface Page
- Incorporating Extensible Flexfields into a Search Form
- Preparing Extensible Flexfield Business Components for Oracle Business Intelligence
- Integrating Extensible Flexfields with Business Intelligence Publisher
- Loading Seed Data
- Customizing the Extensible Flexfield Modelers
- Testing Flexfields
- Making Extensible Flexfields Available to Web Services
- Accessing Information About Extensible Flexfield Business Components
- Using the Advanced Features of Extensible Flexfields.

24.1 Introduction to Extensible Flexfields

An extensible flexfield is similar to a descriptive flexfield in that it provides a customizable expansion space that implementors, such as Oracle Fusion Applications customers, can use to configure additional attributes (segments) without additional programming. As with descriptive flexfields, each segment is represented in the
database as a single column. However, with extensible flexfields, the context values and context-sensitive segments are stored in an extension table.

Implementors can combine and arrange the segments into attribute groups that are tailored to a customer's specific needs. For example, they can group related segments so that they appear together on the page. An attribute group is referred to as a context. You can optionally set up an extensible flexfield to enable implementors to group contexts into categories.

For information about implementors use contexts and categories to configure extensible flexfields, do the following:

• Access the Oracle Fusion Applications Technology library.
• See the Implementing Common Features guides for your product family. Search for flexfields.

To learn more about flexfield basics and terms, including developer and implementor roles, segments, and contexts, see Getting Started with Flexfields. For more details about the differences between descriptive flexfields and extensible flexfields, see The Benefits of Extensible Flexfields.

24.1.1 Understanding Extensible Flexfields

Extensible flexfields include the following key artifacts:

• Contexts (attribute groups)
• Context-sensitive segments
• Logical pages
• Categories
• Category hierarchies
• Usages

24.1.1.1 About Contexts (Attribute Groups)

An extensible flexfield context is a data grouping mechanism that implementors can use to arrange segments into meaningful groups. Each context is a group of attributes that is displayed in its own subregion of the user interface (UI) page at runtime. Implementors create and configure the contexts. After creating a context, an implementor must associate the context with the categories for which that group of attributes is relevant. For example, a Parts flexfield might have a Fax category and an All-in-One Printers category. A Fax context would be associated with both categories, while a Copy context would be relevant only to the All-in-One Printers category. You can learn about categories in About Categories.

The following figure shows the UI page for the Positions business object. The Positions flexfield is embedded in the Additional Position Details region on the page. This region contains the Educational Requirements, Certification and License Requirements, and Travel contexts for the Positions flexfield.
Figure 24-1  Position User Interface Page

Figure 24-2 and Figure 24-3 show UI pages for the Parts business object. The developer has enabled multiple categories for the Parts flexfield, so the page displays the category to which the part shown belongs. In Figure 24-2, the part belongs to the All-in-One Printers category, and in Figure 24-2, the part belongs to the Fax category. In these pages, the Parts flexfield is embedded in the Additional Information region. For the All-in-One Printers category in Figure 24-2, the Additional Information region contains the Copy, Fax, and Scan contexts for the Parts flexfield, while on the Parts page for the Fax category in Figure 24-3, the region contains just the Fax context.

Note:
A flexfield region is empty until the implementor configures the extensible flexfield that is embedded in it.

Extensible flexfields allow implementors to configure contexts as either single row or multiple row. That is, either one set of segments is stored for a business object instance, or multiple sets of segments are stored for the instance. For example, a job position requires only one set of educational requirements, but can require more than one certificate or license.

For single-row contexts, the segments appear as fields in a form. With multiple-row contexts, the segments appear as columns in a table, thus allowing end users to capture a list. In Figure 24-1, Educational Requirements and Travel are single-row contexts, so end users can specify only a single value for each context-sensitive segment, such as the percent of travel time required for the position. The Certification and License Requirements context is a multiple-row context, so end users can enter multiple rows, one for each certificate or license required for the position.
24.1.1.2 Context-Sensitive Segments

With extensible flexfields, every segment is a member of a context (attribute group). That is, every segment is a context-sensitive segment. Implementors define the contexts and their context-sensitive segments. Context-sensitive segments are the lowest-level data points that implementors can define for an extensible flexfield, and each segment is mapped to a column in an extension table. Just as with descriptive flexfields, the segments are rendered as ADF Faces rich client components, such as text box, text area, list of values (LOV) choice list, date picker, checkbox, and radio button group.

In Figure 24-1, the Educational Requirements context contains the High School, Bachelor, Master, J.D., M.D., and Ph.D. context-sensitive segments, and the Certification and License Requirements context contains the Type and Certificate/License context-sensitive segments.

24.1.1.3 About Logical Pages

Extensible flexfields enable implementors to define logical pages with which to group contexts for display purposes. Each page can contain one or more contexts along with their respective context-sensitive segments. There is no limit to the number of contexts
on a logical page. The implementors associate the logical pages with categories on a flexfield usage basis. For information about usages, see About Usages (Data Levels).

In the Parts UI page shown in Figure 24-2, an implementor has defined two logical pages for this category — Printers and All-in-One. The implementor defined the Printers logical page to contain all contexts related to the printing capabilities of the printer, and the All-in-One page to contain all contexts related to the other capabilities of the printer, such as the scanning, copying, and faxing capabilities.

The Parts page that is shown in Figure 24-2 displays the list of logical pages for the category in the left-hand pane. End users view a logical page in the right-hand pane by selecting it from the list. In this figure, the end user has chosen to view the All-in-One logical page. The Printers logical page is shown in the following figure.

**Figure 24-4  Printers Logical Page in the Parts User Interface Page**

Logical pages have several layout options. For example, in Figure 24-1, the developers chose to hide the entire left-hand section because their customers will add attributes to a single page that is known at the time of development. However, in Figure 24-4, developers chose to display the list of logical pages in the left-hand pane. For more information about layout options, see Employing an Extensible Flexfield on a User Interface Page.

### 24.1.1.4 About Categories

Categories enable applications to dynamically display different sets of logical pages and contexts at runtime. In the simplest case, you create a single category for a flexfield, and the same extensible flexfield contexts and logical pages appear for every instance of the business object. In some cases, you might need the application to display different sets of logical pages and contexts depending on a runtime discriminator, such as an instance value. In these cases, you can create multiple categories for the flexfield, or you can provide application-specific logic that enables implementors to define their own categories.

The Positions flexfield in Figure 24-1 has a single category. The same flexfield segments appear in the Additional Position Details region regardless of which position is displayed. In this page, the developer has hidden the name of the category, because there is only one category.
In Figure 24-2 and Figure 24-3, the segments that the region displays depend on which category the part belongs to. In Figure 24-2, the part belongs to the All-in-One Printers category and the UI displays fields related to copying, faxing, and scanning. In Figure 24-3, the part belongs to the Fax category and the UI displays fields related to only faxing.

24.1.1.5 About Category Hierarchies

You can choose to support multiple categories for your flexfield. When you do so, the extensible flexfield respects any hierarchical relationship that is defined for the categories, whether created by developers or defined by implementors. The following figure shows an example of categories that are defined in a hierarchical fashion. In this example, the root category (the category that does not have parent categories) is Computers and Office. One of its child categories is Printers and Ink. The Printers and Ink category, in turn, has a Printers child category. The Printers category has two child categories — All-in-One Printers and Single Function Printers.

![Figure 24-5 Example of a Category Hierarchy](image)

Each child category inherits the contexts and pages from its parent categories. For example, the Printers category contains the Print, Printer Functions, and Supported Operating Systems contexts. The All-in-One Printers category inherits these contexts from the Printers category. It also inherits all logical pages that are defined for its parent categories. Additional contexts can be assigned directly to the All-in-One Printers category, such as Copy and Scan, which are relevant only to All-in-One printers.

24.1.1.6 About Usages (Data Levels)

When you create an extensible flexfield, you create one flexfield usage for each set of tables in the application that uses the flexfield. In the simplest case, you will have one flexfield usage. For example, in Figure 24-1, the Positions application requires a single Positions table with its associated extension tables to store the extensible flexfield values. Implementors simply associate their contexts with that one usage to make them available to the Positions application.

When you have several objects in the application that should be extended using the same extensible flexfield, you must create multiple usages for the flexfield. For example, in the case of an Items application, there might be different data levels, such as items and item revisions. In this case, you create one usage per data level. By
defining separate usages for each set of tables, you enable your implementors to reuse the same extensible flexfield configuration for all data levels.

24.1.2 The Benefits of Extensible Flexfields

When deciding whether to use a descriptive flexfield or an extensible flexfield to extend a business object, consider the following features that you make available for implementors by using extensible flexfields:

- **Custom grouping**: Extensible flexfields enable implementors to group segments into contexts (attribute groups), and each context is displayed in its own region on a page.

- **Multiple logical pages**: When an implementor defines a large number of custom attributes, the implementor can choose to create several logical pages with which to display the attribute groupings. For example, Figure 24-2 and Figure 24-4 show different logical pages for the same part. In addition, the implementors can reuse these pages with different categories.

- **Hierarchical categories**: Extensible flexfields can be configured to enable categories, which can be used to dynamically display different sets of logical pages and contexts based upon a runtime discriminator. The categories can be structured in a hierarchical manner and the children categories inherit all the contexts and logical pages that are configured for their parent categories.

- **Reusable attribute groups**: When you enable multiple categories for an extensible flexfield, implementors can associate contexts with more than one category. When business object instances share common attributes, the implementors can put the common attributes in one context and reuse that context in all applicable categories, thus minimizing setup time. For example, in Figure 24-2 and Figure 24-3, the Fax context is associated with both the Fax category and the All-in-One Printers category.

- **Unlimited expansion space**: While the number of segments that an implementor can define for a context is limited to the number of underlying flexfield extension columns, there is no limit to the number of contexts that an implementor can create for an extensible flexfield. Implementors can use the extension columns over and over in newly created contexts resulting in an unlimited number of custom attributes.

- **Custom lists**: When implementors need to add custom lists to their business objects, they can create multiple-row contexts, which display the context’s segments in a table, as shown in the Certification and License Requirements region in Figure 24-1.

- **Locale-specific values**: You can optionally enable implementors to store different attribute values on a locale-by-locale basis. For example, an implementor could store and maintain the description of a part in English, Chinese, and French. The application displays and stores the value for the end user’s locale.

For more information, see the "Translating Flexfield and Value Set Configurations" section in the Extensibility Guide for Developers.

- **Access control**: Implementors can define who can view and edit the attributes in a context.

- **Unit of measure specifications**: Implementors can define the unit of measure for an attribute.
• Interface tables: Extensible flexfields support the use of interface tables for loading extensible flexfield data.

• Parallel Sets: Extensible flexfields can generate parallel sets of extensible flexfield artifacts for different uses, such as public view objects and private view objects.

Another advantage of using extensible flexfields is that you can create your own custom modeler classes for adding application-specific logic for generating ADF Business Components and UI task flows, as described in Customizing the Extensible Flexfield Modelers.

24.1.3 Extensible Flexfield Structure and Content

Unlike descriptive flexfields, which are mapped to extension columns in a product table, extensible flexfields are mapped to extension columns in extension tables that are separate from the product table. You create the extension tables at design time.

How Extensible Flexfields Are Modeled in Oracle Application Development Framework

Extensible flexfields are modeled as a collection of Oracle Application Development Framework (ADF) polymorphic view rows, as described in the Working with Polymorphic View Rows section in the Developing Fusion Web Applications with Oracle Application Development Framework.

The category hierarchy translates into a hierarchy of view objects that are configured to support polymorphic view rows. The view objects have view links to associated context view objects. A CategoryCode attribute acts as the discriminator that determines which category view row type is used. At application runtime, the category in the base category view object row determines what pages and contexts are displayed. Given a collection of polymorphic view rows, each row can be a different type depending upon the category. An application module that holds the category view object is generated for each category.

Note:

Because flexfield view objects are modeled as polymorphic view objects, you can use extensible flexfield view objects in the same manner that you use any other polymorphic view objects, and they will behave in the same way.

24.2 Overview of Integrating Extensible Flexfields in an Application

The process of developing an extensible flexfield and integrating it in an application comprises several different activities, such as creating the extension tables, creating business components, and adding the flexfield to the appropriate application pages. This section identifies the major tasks for incorporating an extensible flexfield and points to the sections that provide the details for completing the tasks.

Before you begin:

Before you begin developing an extensible flexfield, you must first complete the following tasks:

1. Identify the business object and related product table for which you are implementing the extensible flexfield. For example, you might want to enable implementors to create custom attributes for the Parts business object, which corresponds to the FND_CRM_PARTS table.
2. Decide if you want customers to be able to store translated values for one or more
   segments. For example, for a Parts flexfield, customers might want to store
   translations for the attributes in the Catalog Information context so they can print
   the catalog in different languages.

3. Decide if you want to enable customers to store the information for a context at
different levels (also referred to as flexfield usages). For example, implementors
might choose to set up a context for a Parts flexfield that has both a part supplier
level and a part level. The same context segments appear in both the supplier UI
and the part UI, but the end user would enter different values. Take, for example,
a lead-time segment. At the part level, the end user would supply the average
lead time. At the supplier level, the end user would enter the actual lead time
required for that supplier.

4. Decide if you need more than one set of artifacts generated for a flexfield usage.
   For example, you might need both private user interfaces that are based on
   updatable (entity-based) view objects and public user interfaces that are based on
   read-only view objects. When you have more than one set of artifacts, you will
   need to define multiple groups at the entity-usage level for the flexfield usage,
such as a Private group and a Public group, when you register the flexfield’s
   business components.

5. Determine whether you need to customize the generated model to add additional
   product-specific logic or customize the generated user interface, or both.

To complete the development tasks for an extensible flexfield:

1. Create a base extension table for each flexfield usage. If you want to enable
   customers to store translations on a locale-by-locale basis, create a translation
   extension table and a view of the translation extension table for each usage as
   well.

   See Creating Extensible Flexfield Data Tables.

2. Register the extension tables as secured objects to enable implementors to specify
   view and edit privileges for each context’s attributes.

   See Registering Extension Tables as Secured Objects.

3. Create entity objects for the flexfield’s extension tables. In this step, you create an
   entity object and a view object from the base extension table, and, if they exist, the
   translation extension table and view.

   See Creating and Configuring Extensible Flexfield Entity Objects.

4. Create view objects for the extension tables, create a view object to support
categories, and create a view object to support searching.

   See Creating and Configuring Extensible Flexfield View Objects.

5. Create the flexfield metadata.

   See Defining and Registering Extensible Flexfields .

6. Create and configure business components to support the extensible flexfield.


7. Add the extensible flexfield to the appropriate UI pages.

   See Employing an Extensible Flexfield on a User Interface Page.
8. Optionally, prepare the flexfield for business intelligence.
   See Preparing Extensible Flexfield Business Components for.

9. Load any necessary application seed data, such as error messages and lookup values.
   See Loading Seed Data.

10. Optionally, customize the generated model or the generated UI artifacts, or both.
    See Customizing the Extensible Flexfield Modelers and How to Customize the Runtime User Interface Modeler for Extensible Flexfields.

11. Test the flexfield.
    See Testing Flexfields.

After you have completed the extensible flexfield development process and delivered your application, implementors can use the Manage Extensible Flexfields task flow to define contexts, categories, and pages, and to configure the segments for each extensible flexfield. These task flows determine how the flexfield's segments will be populated, organized, and made available to end users within the application.

For information about planning and implementing flexfield configuration, do the following:

- Access the Oracle Fusion Applications Technology library.
- See the Implementing Common Features guides for your product family. Search for flexfields.

Note:
An extensible flexfield is not displayed at runtime unless at least one context and context-sensitive segment has been configured and associated with a category.

To make the Manage Extensible Flexfields task flow available to application implementors, register it with Oracle Fusion Functional Setup Manager. For more information, see Integrating Flexfield Task Flows into.

### 24.3 Create Extensible Flexfield Data Tables

Before you can define an extensible flexfield, you must create one set of dedicated database tables for each of the flexfield's usages. Each set must be composed of at least a base extension table. If you want to enable customers to store translated values for the segments, the set must include a translation extension table and a view of the translation extension table.

At runtime, as end users enter attribute values for each context, the context and its segment (attribute) values are stored as a row in the base extension table if the context is not translatable, or, if the context is translatable, as rows in the translation extension table (one row per installed language). The translation extension view returns the rows in the current user locale. When an implementor creates a translatable context, the implementor can add only VARCHAR2 segments to the context.

An extensible flexfield must have at least one set of tables, which defines the primary usage. The implementors will expose only the extension columns that they require.
You must use the Database Schema Deployment Framework tools to create the tables and columns. Using these tools ensures that the table and its columns are registered in the Oracle Fusion Middleware Extensions for Applications (Applications Core) data dictionary. For more information, see Using the Database Schema Deployment Framework.

Note:
The steps in this section assume that the product table that the flexfield is extending already exists.

24.3.1 How to Create a Base Extension Table

Each flexfield usage requires a base extension table. This table stores nontranslatable contexts and their segment values. The table name should have a suffix of _B to identify it as the base extension table. The name should have the _B suffix regardless of whether the extensible flexfield has a translation table.

Table 24-1 lists the columns that you must include in a base extension table. You can include additional columns, such as columns that are required by product teams or application standards. For space considerations, the table shows only the first and last attribute columns for each type.

Add as many attribute columns as necessary. The attribute columns can be of type VARCHAR2, NUMBER, and TIMESTAMP. Do not set scale or precision for the NUMBER columns. There are no size requirements for the VARCHAR2 columns. To avoid compatibility and interoperability problems, name the columns ATTRIBUTE_type-and-number, such as ATTRIBUTE_CHAR1 for a VARCHAR2 column, ATTRIBUTE_NUMBER1 for a NUMBER column, and ATTRIBUTE_TIMESTAMP1 for a TIMESTAMP column.

For columns of type NUMBER, you can optionally add a column named ATTRIBUTE_NUMBERn_UOM to store the unit of measure.

The primary key of the base extension table must include the EFF_LINE_ID column, the CONTEXT_CODE column, the CATEGORY_CODE column, and the columns that match the primary key columns of the product table.

Table 24-1  Extensible Flexfield Base Extension Table (_B) Specification

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Nullable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFF_LINE_ID</td>
<td>NUMBER (18)</td>
<td>No</td>
</tr>
<tr>
<td>Primary key columns of the product table for which this extensible flexfield is being defined.</td>
<td>Same as the column in the application.</td>
<td>No</td>
</tr>
<tr>
<td>CONTEXT_CODE</td>
<td>VARCHAR2 (80)</td>
<td>No</td>
</tr>
<tr>
<td>CATEGORY_CODE</td>
<td>VARCHAR2 (80)</td>
<td>No</td>
</tr>
<tr>
<td>CREATED_BY</td>
<td>VARCHAR2 (64)</td>
<td>No</td>
</tr>
<tr>
<td>CREATION_DATE</td>
<td>TIMESTAMP</td>
<td>No</td>
</tr>
<tr>
<td>LAST_UPDATED_BY</td>
<td>VARCHAR2 (64)</td>
<td>No</td>
</tr>
</tbody>
</table>
### Table 24-1  (Cont.) Extensible Flexfield Base Extension Table (\_B) Specification

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Nullable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAST_UPDATE_DATE</td>
<td>TIMESTAMP</td>
<td>No</td>
</tr>
<tr>
<td>LAST_UPDATE_LOGIN</td>
<td>VARCHAR2 (32)</td>
<td>Yes</td>
</tr>
<tr>
<td>ATTRIBUTE_CHAR1</td>
<td>VARCHAR2 (150)</td>
<td>Yes</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTRIBUTE_CHAR40</td>
<td>VARCHAR2 (150)</td>
<td>Yes</td>
</tr>
<tr>
<td>ATTRIBUTE_NUMBER1</td>
<td>NUMBER</td>
<td>Yes</td>
</tr>
<tr>
<td>ATTRIBUTE_NUMBER1_UOM</td>
<td>VARCHAR2 (9)</td>
<td>Yes</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTRIBUTE_NUMBER20</td>
<td>NUMBER</td>
<td>Yes</td>
</tr>
<tr>
<td>ATTRIBUTE_NUMBER20_UOM</td>
<td>VARCHAR2 (9)</td>
<td>Yes</td>
</tr>
<tr>
<td>ATTRIBUTE_TIMESTAMP1</td>
<td>TIMESTAMP</td>
<td>Yes</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATTRIBUTE_TIMESTAMP10</td>
<td>TIMESTAMP</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### 24.3.2 How to Create a Translation Extension Table

If you plan to allow customers to store translations for some contexts on a locale-by-locale basis, then you must also create a translation extension table and a single table translation view from that table. The translation extension table and the translation extension view are not required. However, if you do not provide these tables, the customers cannot store translations for their extensible flexfields as described in the “Translating Flexfield and Value Set Configurations” section in the *Extensibility Guide for Developers*. For more information about multilanguage tables, see *Understanding Multi-Language Support Features*. For information about creating the view, see *How to Create a Translation Extension View*.

Note that unlike typical translation tables, the base extension table is completely separate from the translation extension table. All the segment values for a context go in either the base extension table or in the translation extension table. If a context is marked as translatable, all segment values are stored in the translation extension table. If a context is not marked as translatable all segment values are stored in the base extension table. This means that a translatable context cannot contain segments of non-translatable data types, such as numbers or dates.

*Table 24-2* lists the required translation extension table columns. The required columns are similar to those in the base extension table, except that the attribute columns must be of type `VARCHAR2` and you must include the `LANGUAGE` and `SOURCE_LANG` columns. For space considerations, the table shows only the first and last attribute columns. You can include additional columns, such as columns that are required by product teams or application standards.
The translation table name should have a suffix of _TL to identify it as a translation table.

The primary key of the translation extension table must include the EFF_LINE_ID column, the CONTEXT_CODE column, the CATEGORY_CODE column, the LANGUAGE column, and the columns that match the primary key columns of the product table.

Developers typically create the same number of VARCHAR2 type attribute columns in the translation extension table as exist in the base extension table, but you are not required to do so. Create as many VARCHAR2 type attribute columns as you think are necessary for a translatable context. To avoid compatibility and interoperability problems, name the columns ATTRIBUTE_CHARnumber, such as ATTRIBUTE_CHAR1, ATTRIBUTE_CHAR2, and so on. When setting the size of the VARCHAR2 columns in an extension translation table, consider that the columns might need to contain multibyte characters.

Table 24-2 Extensible Flexfield Translation Extension Table (_TL) Specification

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Nullable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFF_LINE_ID</td>
<td>NUMBER(18)</td>
<td>No</td>
</tr>
<tr>
<td>Primary key columns of the product table for which this extensible flexfield is being defined.</td>
<td>Same as the column in the application.</td>
<td>No</td>
</tr>
<tr>
<td>CONTEXT_CODE</td>
<td>VARCHAR2(80)</td>
<td>No</td>
</tr>
<tr>
<td>CATEGORY_CODE</td>
<td>VARCHAR2(80)</td>
<td>No</td>
</tr>
<tr>
<td>CREATED_BY</td>
<td>VARCHAR2(64)</td>
<td>No</td>
</tr>
<tr>
<td>CREATION_DATE</td>
<td>TIMESTAMP</td>
<td>No</td>
</tr>
<tr>
<td>LAST_UPDATED_BY</td>
<td>VARCHAR2(64)</td>
<td>No</td>
</tr>
<tr>
<td>LAST_UPDATE_DATE</td>
<td>TIMESTAMP</td>
<td>No</td>
</tr>
<tr>
<td>LAST_UPDATE_LOGIN</td>
<td>VARCHAR2(32)</td>
<td>Yes</td>
</tr>
<tr>
<td>SOURCE_LANG</td>
<td>VARCHAR2(4)</td>
<td>No</td>
</tr>
<tr>
<td>LANGUAGE</td>
<td>VARCHAR2(4)</td>
<td>No</td>
</tr>
<tr>
<td>ATTRIBUTE_CHAR1</td>
<td>VARCHAR2(1000)</td>
<td>Yes</td>
</tr>
<tr>
<td>...</td>
<td>VARCHAR2(1000)</td>
<td>Yes</td>
</tr>
<tr>
<td>ATTRIBUTE_CHAR40</td>
<td>VARCHAR2(1000)</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Note:
To avoid compatibility and interoperability problems, format the column names as indicated.
24.3.3 How to Create a Translation Extension View

If you create a translation extension table, you must also create a view from the translation extension table. To create the view, use a view definition statement that follows the format shown in Example 24-1. The name of the translation extension view should have a suffix of _VL.

Example 24-2 shows a sample view definition.

**Example 24-1 Format for Translation Extension View Definition**

```sql
select ROWID ROW_ID, 
column list
from translation extension table
where LANGUAGE = userenv('LANG');
```

**Example 24-2 Sample Translation Extension View Definition**

```sql
select
ROWID ROW_ID,
EFF_LINE_ID,
CONTEXT_CODE,
CATEGORY_CODE,
ATTRIBUTE_CHAR1,
...
ATTRIBUTE_CHAR40
from MYEFF_TL
where LANGUAGE = userenv('LANG');
```

24.4 Register Extension Tables as Secured Objects

You must register an extensible flexfield’s extension tables as database resources to enable implementors to specify and manage access privileges for context attributes.

You can optionally preconfigure database resource actions and conditions (instance sets) that might be useful for the customers.

---

**Note:**

When an implementor uses Oracle Authorization Policy Manager to define security policies for an extensible flexfield, the implementor must manage the privileges for both the base extension table and the view of the translation extension table. If your application has a separate user interface for managing privileges, this user interface must manage the base extension tables and the views of the translation extension tables for all usages as separate secured objects.

---

24.4.1 How to Register a Table as a Secured Object

You must register an extensible flexfield’s base extension table as a secured object by registering a database resource for the table. If you create a translation extension table and view for an extensible flexfield, then you must register the view as a database resource as well. Use the Manage Database Security Policies task, which is accessed from the Setup and Maintenance work area of any Oracle Fusion Setup application, to create the database resources.
You can optionally create actions and conditions for the database resource. The preconfigured actions appear in the View Privileges and Edit Privileges dropdown lists for a context’s usages in the Manage Extensible Flexfields task. Customers use these actions to restrict view and edit privileges.

To create a database resource:

1. In the Setup and Maintenance work area of any Oracle Fusion Setup application, go to the Manage Database Security Policies task.

2. From the Search Results region, select Actions > Create.

3. Register the database resource.

---

WARNING:

When registering a database resource for the base extension table, you must use the name of the table for the name of the database resource, such as ITEM_EFF_B. When registering a database resource for the view of the translation extension table, you must use the name of the view for the name of the database resource, such as ITEM_EFF_VL. Otherwise, if an implementor creates actions for that database resource and sets the privileges for an extensible flexfield’s context to those actions, the end users who have those privileges will get an oracle.jbo.JboException, such as Object is not registered with Data Security, when they access a page that contains the extensible flexfield context.

---

4. Ensure that the EFF_LINE_ID column is included in the primary key.

5. Optionally, create actions and conditions for the database resource.

**24.5 Create and Configuring Extensible Flexfield Entity Objects**

Before you can register an extensible flexfield, you must create entity objects for the flexfield’s extension tables and you must configure the EFF_LINE_ID attribute for each extensible flexfield entity object as a unique ID.

For information about entity objects, see the Creating a Business Domain Layer Using Entity Objects chapter in the Developing Fusion Web Applications with Oracle Application Development Framework. For information about unique IDs see About Using Unique ID.

---

**Note:**

The packages in which these entity objects are created must not fall under the packages allocated for runtime-generated business components (which are specified in the adf-config.xml file and the flexfield usage metadata).

---

**24.5.1 How to Create and Configure Extensible Flexfield Entity Objects**

Use the Create Entity Object wizard to create and configure an entity object from the base extension table. If a translation extension table exists for the flexfield, create and configure extensible flexfield entity objects from the translation extension table and the translation table view.

**Before you begin:**
1. Create one set of dedicated extension tables per usage, as described in How to Create a Base Extension Table.

2. Ensure that the Applications Core library has been added to the data model project, as described in Adding Necessary Libraries to Your Data Model Project.

3. Verify that at least one customization class is included in the adf-config.xml file. This serves to ensure correct application behavior. It does not matter which customization class you include.

   For information about defining the customization layers, see the "Understanding Customization Layers" section in the Extensibility Guide for Developers.

24.5.1.1 Creating and Configuring an Entity Object from the Base Extension Table

Use the Create Entity Object wizard to create an entity object from the base extension table that is described in Table 24-1, but apply the changes described in the following procedure to support the extensible flexfield. For information about the Create Entity Object wizard, see the How to Create Single Entity Objects Using the Create Entity Wizard section in the Developing Fusion Web Applications with Oracle Application Development Framework.

To create and configure an entity object from the base extension table:

1. On the Attributes page of the wizard, include only the following columns as attributes in your new entity object:
   - EFF_LINE_ID
   - OBJECT_VERSION_NUMBER
   - Primary key columns of the product table
   - CONTEXT_CODE
   - CATEGORY_CODE
   - CREATED_BY
   - CREATION_DATE
   - LAST_UPDATED_BY
   - LAST_UPDATE_DATE
   - LAST_UPDATE_LOGIN

2. On the Attribute Settings page, do the following:
   - Set the EFF_LINE_ID attribute to be a primary key.
   - Set the CONTEXT_CODE attribute to be a discriminator, with a default value of 0.

3. On the Java page, confirm that the objectnameBEOImpl entity object class to be generated extends the oracle.apps.fnd.applcore.oaext.model.OAEntityImpl class.

4. After creating the entity object, click the Attributes navigation tab and complete the following steps:
a. Select the EFF_LINE_ID attribute and, on the Applications tab of the Property Inspector, set Application Unique ID to true to generate a globally unique id.

b. On the UI Hints tab of the Property Inspector for each attribute, set the Display Hint property to Hide

5. Create an alternate key for the flexfield.

When business components are generated from this entity object, this alternate key will be extended to include all multiple-row context segments that implementors designate as a unique key when they configure the flexfield.

**Note:**

If this alternate key does not exist for an extensible flexfield, implementors will not be able to mark a segment in a multiple-row context as a unique key.

---

24.5.1.2 Creating and Configuring an Entity Object from the Translation Extension Table

If you created a translation extension table for the flexfield usage, as described in Table 24-2, use the Create Entity Object wizard to create an entity object from the translation extension table, but apply the changes described in the following procedure to support the extensible flexfield.

For more information about creating entity objects, see the Creating a Business Domain Layer Using Entity Objects chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

To create and configure an entity object from the extensible flexfield translation extension table:

1. On the Attributes page of the wizard, include only the following columns as attributes in your new entity object:
   - EFF_LINE_ID
   - OBJECT_VERSION_NUMBER
   - Primary key columns of the product table
   - CONTEXT_CODE
2. On the Attribute Settings page, do the following:
   - Set the EFF_LINE_ID attribute to be a primary key.
   - Set the CONTEXT_CODE attribute to be a discriminator, with a default value of 0.

3. On the Java page, confirm that the objectnameTlEOImpl entity object class to be generated extends the oracle.apps.fnd.applcore.oaext.model.OAEntityImpl class.

4. After creating the entity object, click the Attributes navigation tab and complete the following steps:
   a. Select the EFF_LINE_ID attribute and, on the Applications tab of the Property Inspector, set Application Unique ID to true to generate a globally unique id.
   b. On the UI Hints tab of the Property Inspector for each attribute, set the Display Hint property to Hide

5. Configure all of the non-key attributes that are of type String to be translatable.
   On the Applications tab of the Property Inspector for each non-key, String type attribute, set the OA Translatable property to True.

6. Create an alternate key for the flexfield.
   When business components are generated from this entity object, this alternate key will be extended to include all multiple-row context segments that implementors designate as a unique key when they configure the flexfield.

Note:
If this alternate key does not exist for an extensible flexfield, implementors will not be able to mark a segment in a multiple-row context as a unique key.

a. In the Application Navigator, double-click the entity object.

b. In the overview editor, click the General tab.

c. In the Alternate Keys section, click the Add icon.
d. In the Define Alternate Key dialog, type EFF_ALT_KEY in the Alternate Key Name field.

e. Select the primary key of the product table that the extensible flexfield extends and move it to the Selected list.

f. Select the contextCode attribute, move it to the Selected list, and click OK.

24.5.1.3 Creating and Configuring an Entity Object from the Translation Extension View

If you created a translation extension table and view for the flexfield usage, use the standard wizard to create an entity object from the translation extension view that is described in How to Create a Translation Extension View, but apply the changes described in the following procedure to support the extensible flexfield.

To create and configure an entity object from the extensible flexfield translation extension view:

1. On the Attributes page of the wizard, include only the following columns as attributes in your new entity object:
   - EFF_LINE_ID
   - Primary key columns of the product table
   - CONTEXT_CODE
   - CATEGORY_CODE

2. On the Attribute Settings page, do the following:
   - Set the EFF_LINE_ID attribute to be a primary key.
   - Set the CONTEXT_CODE attribute to be a discriminator, with a default value of 0.

3. On the Java page, confirm that the objectnameV1EOImpl entity object class to be generated extends the oracle.apps.fnd.applcore.oaext.model.OAEntityImpl class.

4. After creating the entity object, click the Attributes navigation tab and complete the following steps:
   a. Select the EFF_LINE_ID attribute and, on the Applications tab of the Property Inspector, set Application Unique ID to true to generate a globally unique id.
   b. On the UI Hints tab of the Property Inspector for each attribute, set the Display Hint property to Hide

5. On the Applications tab of the Property Inspector for the entity object General tab, set the OA Base Table property to the name of the view underlying this entity object.
24.6 Create and Configuring Extensible Flexfield View Objects

After you create and configure entity objects from the base extension table, create and configure flexfield view objects to support contexts, categories, and, optionally, searching.

24.6.1 How to Create and Configure Extensible Flexfield View Objects

Use the Create View Object wizard (not the Flexfield Business Components wizard) to create the following types of view objects:

- To support contexts for the extensible flexfield, create one view object from the base extension table entity object and, if the flexfield has a translation extension table, one view object from the translation extension view entity object.

  **Note:**
  No view objects are required from the translation table entity object.

- To support categories for the flexfield, create a view object from the product table entity object for which you are developing this extensible flexfield.

- To support searching, create a declarative SQL-based view object.

For more information about creating view objects, see the Defining SQL Queries Using View Objects chapter in the Developing Fusion Web Applications with Oracle Application Development Framework. For more information about declarative SQL-based view objects, see the How to Create Declarative SQL View Objects section in the Developing Fusion Web Applications with Oracle Application Development Framework.

Before you begin:

1. Create and configure entity objects from the base extension table, translation extension table, and view as described in How to Create and Configure Extensible Flexfield Entity Objects.

24.6.1.1 Creating and Configuring Context View Objects

To support contexts for an extensible flexfield usage, the usage must have a view object from its base extension table entity object. If you created a translation extension table for the usage, you must also create a view object from the translation extension view entity object.

To create and configure context view objects:

1. Use the Create View Object wizard to create the view objects from the base extension table entity object and, if you created a translation extension table for the usage, from the translation extension view entity object.

2. On the Java page for each view object, ensure that the view object classes to be generated extend base classes as follows:

   - `vo_nameImpl extends EFFViewObjectImpl`
   - `vo_nameRowImpl extends EFFViewRowImpl`
• `vo_nameDefImpl extends EFFViewDefImpl`

These base classes are in the `oracle.apps.fnd.applcore.oaext.model` package.

### 24.6.1.2 Creating and Configuring the Base Category View Object

To support categories for the flexfield, there must be a view object for the product table entity object for which you are developing the extensible flexfield.

**To create and configure the base category view object:**

Use the Create View Object wizard to create a view object from the product table for which you are implementing the extensible flexfield in the usual manner. For example, in Figure 24-2, the product table is the Parts table that stores the base attributes for the Parts entity, such as Part Number, Description, and Unit Cost.

This view object must include an attribute called `CategoryCode`, which is used to identify the category to which each row of data belongs. This attribute must be an entity-based attribute of type `VARCHAR2` with a maximum of 80 characters.

Set the `CategoryCode` attribute to be a discriminator, with a default value of 0.

You must ensure that the correct category code is returned for this attribute at runtime.

### 24.6.1.3 Create a Declarative SQL-Based View Object to Enable Searching

To enable searching or to enable business intelligence, you must create a declarative SQL-based view object for searching the product table for which you are implementing the extensible flexfield. You must include the same product table entity object that was used for the base category view object. For more information about the base category view object, see [Creating and Configuring the Base Category View Object](#).

For information about declarative SQL-based views objects, see the *How to Create Declarative SQL View Objects* section in the *Developing Fusion Web Applications with Oracle Application Development Framework*.

**To create a declarative SQL-based view object to enable searching:**

1. In the Application Navigator, right-click the project in which you want to create the view object and choose New.

2. In the New Gallery, expand Business Tier, select ADF Business Components and then View Object, and click OK.

3. On the Name page, enter a package name and a view object name. Keep the default setting **Updatable access through entity objects** enabled to indicate that you want this view object to manage data with its base entity object. Click Next.

4. On the Entity Objects page, select the entity object that was used for the base category view object and move it to the Selected tree.

   The view object can contain additional entity objects, but it must include the entity object that was used for the base category view object.

5. Click Next.
6. On the Attributes page, select all the attributes for which you want to enable search and move them to the Selected tree.
   You must include the CategoryCode attribute and the primary key attributes.

7. Select the other attributes to include in the view criteria, move them to the Selected tree, and click Next.

8. Mark the CategoryCode attribute as a discriminator with a default value of −1.
   a. On the Attribute Settings page, select CategoryCode from the Select Attribute dropdown list.
   b. Select Discriminator.
   c. In the Default Value field, type −1.

   Note:
   CategoryCode must be an entity object attribute. It cannot be a transient attribute.

9. Configure the attribute settings for the remaining attributes as required, and click Next.

10. On the Query page, select Declarative in the SQL Mode dropdown list if it is not already displayed. The wizard changes to declarative SQL mode.

11. On the Java page, select Generate View Object Class and select Generate View Object Definition Class.

12. Click Classes Extend.

13. In Object field in the Override Base Classes dialog, type oracle.apps.fnd.applcore.oaext.model.EFFCategoryViewObjectImpl.
    The EFFCategoryViewObjectImpl class adds the additional WHERE clause that provides security.

14. In the Definition field, type oracle.apps.fnd.applcore.oaext.model.FlexfieldBaseViewDefImpl, and click OK.

15. Complete the steps.

16. Either edit the flexfield in the Register Flexfield Metadata wizard and provide the fully qualified name of the search view object or execute the following SQL statement to register it. Replace search-view-object with the fully qualified name for the search view object. Replace product-table with the name of the product table that the flexfield extends.

    ```sql
    UPDATE FND_DF_ADFBC_USAGES
    SET EFF_SEARCH_VIEW_OBJECT = 'search-view-object'
    WHERE TABLE_NAME='product-table'
    AND ENTITY_OBJECT='product-table-entity-object';
    ```
24.7 Defining and Registering Extensible Flexfields

After you create the flexfield’s extension tables and their entity objects and view objects, you can define and register the extensible flexfield’s usage. The definition of an extensible flexfield usage includes the following information:

- The flexfield’s code, name, and description.
- The flexfield usage’s code, name, and description.
- The name of the product table for which the flexfield is implemented (the primary table), and the view object and application module for that table.
- The name of the base extension table that contains the columns to be used as flexfield segments. This name should have a suffix of _B.
- The name of the context column in the base extension table. This should be CONTEXT_CODE.
- The names of the columns in the base extension table that are to be used as flexfield segments.
- If the flexfield is enabled for translations, the name of the translation extension table, which should have a suffix of _TL, and the translation extension view, which should have a suffix of _VL.
- Flexfield ADF Business Component usage information, including the Oracle Metadata Services (MDS) root packages for its usage — the name of the root package for all business components that model the flexfield. Each usage must have its own package name.
- If the flexfield has custom modelers, the names of the custom modeler. For information about custom modelers, see How to Customize the Runtime Business Component Modeler for Extensible Flexfields and How to Customize the Runtime User Interface Modeler for Extensible Flexfields.

After the implementors configure the flexfield, the definition of an extensible flexfield also contains the following information:

- A complete list of the ContextCode values that can appear in the flexfield context segment.
- Information about the segments that are associated with the ContextCode values. Each ContextCode value is associated with its own set of these segments.
- A complete list of the CategoryCode values that belong to the extensible flexfield, and the contexts that are associated with each category, page, and usage.
- Validation rules that are associated with the segments. Each segment can have its own LOV or validation rules.

24.7.1 How to Register Extensible Flexfields

You can use either of the of the following methods to define and register an extensible flexfield usage:

- Flexfield Registration Metadata wizard
24.7.1.1 Using the Flexfield Registration Metadata Wizard to Register and Define Extensible Flexfields

Use the **Register new flexfield** operation to register and define an extensible flexfield from the Flexfield Registration Metadata wizard. To make subsequent additions and changes to the flexfields, you use the **Edit flexfield operation**.

**Before you begin:**

- Create the extensible flexfield tables as described in *Creating Extensible Flexfield Data Tables*.
- Create entity objects for the extensible flexfield tables as described in *Creating and Configuring Extensible Flexfield Entity Objects*.
- Create view objects for the extensible flexfield tables as described in *Creating and Configuring Extensible Flexfield View Objects*.
- Ensure that entity objects exist for the primary table (the product table that the extensible flexfield extends).

**To register and define an extensible flexfield using the Flexfield Registration Metadata wizard:**

1. From the **File** menu, choose **New**.
2. In the New Gallery, expand **Business Tier**, select **ADF Business Components** and then **Flexfield Registration Metadata**, and click **OK**.
3. On the Choose Operation page of the Flexfield Registration Metadata wizard, select **Register new flexfield** as shown in the following figure, and then click **Next**.
4. On the Basic Information page, select **Extensible Flexfield** from the *Type* dropdown list.

5. Provide the following values:
   - **Code**: Type a code that uniquely identifies the flexfield.
   - **Name**: Type a descriptive name for the flexfield.
   - **Description**: Type a short description of the flexfield.
   - **Application**: Select the application name that has been assigned for the product area.
   - **Module**: Select the module that owns the flexfield. This is typically the application or logical business area (LBA) with which the flexfield is delivered.

6. Click **Next**.

7. On the Product Table page, set the following values to define the flexfield usage.
   - Flexfield usage code, name, and description.
   - The name of the product table for which the flexfield is implemented, and the view object and application module for the product table.
   - Provide the search view object, if it exists. This is an SQL-based view object that is based on the entity object that was used for the base category view object as described in **Creating a Declarative SQL-Based View Object to Enable Searching**.
Note:
You need a search view object only if the flexfield is enabled for business intelligence or there is a need to expose the flexfield’s segments in the search user interface for the product table that the flexfield is extending, as described in Incorporating Extensible Flexfields into a Search Form.

- The name of the root package to be used for the generated business components that model the flexfield usage.
- The object name prefix for the flexfield usage. For example, PartsEFF. This prefix is used to derive the names of objects that are generated for the flexfield usage.

8. Click Next.

9. From the Extension Table page, specify the name of the base extension table for the flexfield, and the table’s entity object and view object.

10. Select the database columns in the Available list that you want to use for the flexfield segments and move them to the Selected list.

The wizard assumes that the CATEGORY_CODE column is to be used for the category code and that the CONTEXT_CODE column is to be used for the context code.

11. Click Next.

12. On the Extension TL Table page, complete the following steps if a translation extension table exists for the flexfield:
   a. Select Enable translatable attributes.
   b. Provide the name of the translation extension table.
   c. Provide the names of the entity object and view object for the translation extension table, and the name of the entity object for the translation extension view. These are the objects that you created in How to Create and Configure Extensible Flexfield Entity Objects.
   d. In the Available list in the Columns region, move the CONTEXT_CODE column to the Selected list.
   e. Select the database columns in the Available list that you want to map to flexfield segments and move them to the Selected list.

13. Click Next.

14. On the Advanced Options page, enter the name of the flexfield’s custom business component modeler and custom user interface modeler if you have created these modelers for the flexfield.

15. Click Next.

16. On the Summary page, optionally select Launch Flexfield Business Components Wizard if you are ready to create business components for the flexfield.

17. Click Finish.
18. Add entries to the FND_EF_CATEGORIES_B and FND_EF_CATEGORIES_TL tables for the root category code. In the FND_EF_CATEGORIES_B table, set the PARENT_CATEGORY_CODE to null to indicate that it is the root category. All other category rows for this column points to the root or its child categories.

An extensible flexfield requires at most, one root category, and at least one entry in the FND_EF_CATEGORIES_B table, and, if one exists, the FND_EF_CATEGORIES_TL table. Add additional entries only if you are shipping your application with some categories already defined.

The extensible flexfield category’s schema definition object, or XSD file, is included in its parent category’s XSD file. Thus, the parent category can load the schema for any child category.

---

**Note:**

The CATEGORY_CODE and PARENT_CATEGORY_CODE values are used in the flexfield’s element in the XML schema for web services. You can maximize the readability of the schema by naming the codes with a leading alphabetic character followed by alphanumeric characters. The use of spaces, underscores, multibyte characters, and leading numeric characters, which are all encoded in XML schemas, make the codes in the schema element difficult to read.

---

### 24.7.1.2 Using the Setup APIs to Register and Define Extensible Flexfields

In addition to using the Flexfield Registration Metadata wizard, you can use the FND_FLEX_EF_SETUP_APIS PL/SQL API package to register new extensible flexfield usages and to add the flexfield metadata.

You can learn about the FND_FLEX_EF_SETUP_APIS PL/SQL package by running the following command, which stores package documentation and usage examples in the `<db_name>_<user_name>_FND_FLEX_EF_SETUP_APIS_<date>.plsqldoc` file.

```
sqlplus <fusion_user>/<fusion_pwd>@<fusion_db> \
@/ORACLE/fusionapps/atgpf/applcore/db/sql/flex/fnd_flex_pkg_doc.sql \
FND_FLEX_EF_SETUP_APIS
```

After you register the flexfield, you must add entries for the root category code to the FND_EF_CATEGORIES_B and FND_EF_CATEGORIES_TL tables. Set the PARENT_CATEGORY_CODE to null to indicate that it is the root category.

An extensible flexfield requires at least one entry in the FND_EF_CATEGORIES_B table and, if one exists, the FND_CATEGORIES_TL table. Add additional entries only if you are shipping your application with some categories already defined.

---

**Note:**

The CATEGORY_CODE and PARENT_CATEGORY_CODE values are used in the flexfield’s element in the XML schema for web services. You can maximize the readability of the schema by naming the codes with a leading alphabetic character followed by alphanumeric characters. The use of spaces, underscores, multibyte characters, and leading numeric characters, which are all encoded in XML schemas, make the codes in the schema element difficult to read.
24.8 Creating Extensible Flexfield Business Components

To incorporate an extensible flexfield into your application, you must create business components for the flexfield usage.

The following figure shows the extensible flexfield business components for a product table. In this example, the flexfield is not translatable. The developer has created an entity object and view object from the base extension table to support contexts.

When an implementor deploys flexfield configurations, the deployment process creates the extended context entity objects, context view objects, category view objects, and category-to-context view links. Note that in this example, the C2 context is a member of two categories.
Creating Extensible Flexfield Business Components

**Figure 24-7 Extensible Flexfield Business Components**
24.8.1 How to Create Business Components For an Extensible Flexfield's Usage

You use the Create Flexfield Business Components wizard to create the flexfield business components for an extensible flexfield’s usage.

The business components that are generated will replace any existing components that are based on the same flexfield usage.

**Tip:**

After completing these steps, you can regenerate the flexfield business components programmatically at runtime to update your extensible flexfield implementation without manual intervention.

For more information, see Regenerating Flexfield Business Components Programmatically.

**Before you begin:**

- Create the flexfield entity objects as described in How to Create and Configure Extensible Flexfield Entity Objects.
- Create the flexfield view objects and the declarative SQL-based view object as described How to Create and Configure Extensible Flexfield View Objects.
- Register the flexfield as described in How to Register Extensible Flexfields.
- Build your project to ensure that the entity objects are available in classes. The modeler relies on what is in your classes.
- Ensure that you have added the Applications Core library to your project as described in Setting Up Your JDeveloper Application Workspace and Projects.
- Ensure that at least one customization class is included in the adf-config.xml file. This inclusion serves to ensure correct application behavior. It does not matter which customization class you include.

For information about customization layers, see the "Understanding Customization Layers" section in the Extensibility Guide for Developers.

**To create extensible flexfield business components:**

1. From the File menu, choose New.

2. In the New Gallery, expand Business Tier, select ADF Business Components, select Flexfield Business Components, and click OK.

3. In the Flexfield page of the Create Flexfield Components wizard, select Extensible from the Type dropdown list as shown in the following figure.
4. In the **Application** field, specify the full name of the application to which your extensible flexfield belongs.

   You can browse for the name, and filter by **ID**, **Short Name**, or **Name**.

5. In the **Code** field, specify the code of the extensible flexfield you want to use.

   You can browse for and filter by **Code**.

6. In the **Usage** section, select the table row that contains your desired extensible flexfield usage.

7. Click **Next**.

8. The Overview page displays the tasks that must be completed before the wizard can generate the business components, as shown in the following figure. The wizard will complete the necessary tasks. Click **Next**.

**Note:**

If you created a declarative SQL-based view object as described in **Creating a Declarative SQL-Based View Object to Enable Searching**, then the wizard will create a stub search view object, which you can use to expose the extensible flexfield’s attributes in a table and to incorporate the extensible flexfield in a search form. Make a note of the fully qualified path that is shown in on the Overview page for the stub search view object so that you can locate the view object and add it to an application module. If a path is not shown for the view object, then the declarative SQL-based view object was not registered in the FND_DF_ADFBC_USAGES table.
9. If the wizard needs to implement any uncompleted tasks from the Overview page, the wizard displays the automated code changes, as shown in the following figure. Review the code and click Next.


Note:
This wizard might fail with a "ClassNotFoundException" exception message. This indicates that one or more required libraries have not been automatically included in your project. You can resolve this issue by manually adding any missing libraries; then you can complete this procedure successfully.
11. Refresh the project to see the newly created flexfield business components in the Application Navigator. The components are in the package that you registered for the flexfield usage and are named using the registered prefix.

24.9 Employ an Extensible Flexfield on a User Interface Page

You can incorporate an extensible flexfield into an application with several UI variations:

- As a single task flow for multiple contexts for a single usage in one of the following layouts:
  - Single page layout: A single task flow that presents a set of contexts associated with a specified logical page, which is identified before the UI page initializes. Figure 24-1 is an example of this variation.
  - Panel splitter layout: A list of the extensible flexfield logical pages for a single usage, combined with the contexts for the selected logical page, and integrated into a single task flow. Figure 24-2 is an example of this variation.

- As a single task flow that presents a context, which is identified before the UI page initializes.

- As a complete list of the usages and pages for a given extensible flexfield in one task flow, with contexts for the selected page in a separate task flow.

- As columns in a table that displays the attributes for a specific category. The attributes from all the contexts that are associated with the category are displayed. The attributes are grouped in a column span and are labeled *context*:*segment*.

Note:

- If there is only one usage and only one logical page, use a single task flow that presents a set of contexts associated with a specified logical page, which is identified before the UI page initializes. Do not display a list of usages or pages on the left-hand side of the task flow.

- If the extensible flexfield will never have more than one usage and will never have more than one logical page, and if you do not need any features that are unique to extensible flexfields as discussed in *The Benefits of Extensible Flexfields*, consider using a descriptive flexfield instead.

- Unless you know that the extensible flexfield will contain only a small amount of information, do not embed the flexfield in the middle of a region that has other content.

For more information about task flows, see the Creating a Task Flow section in the *Developing Fusion Web Applications with Oracle Application Development Framework*. 
24.9.1 How to Expose an Extensible Flexfield Usage as a Single Task Flow for Multiple Contexts

You can easily incorporate an extensible flexfield into the UI in one of the following layouts by dragging the view object for the flexfield-enabled object from the Data Controls panel and dropping it onto the page:

- Single page layout: The extensible flexfield is incorporated as a single task flow that presents a set of contexts associated with a specified logical page and usage as shown in Figure 24-1.

- Panel splitter layout: The extensible flexfield is incorporated as a list of the logical pages for a specified usage, combined with the contexts for the user-selected logical page, and integrated into a single task flow, as shown in Figure 24-2.

Before you begin:

1. Create and configure business components to support the extensible flexfield as described in Creating Extensible Flexfield Business Components.

2. Execute the following SQL query to obtain the name of the view object for the flexfield-enabled object:

   ```sql
   SELECT VIEW_OBJECT FROM FND_DF_ADFBC_USAGES WHERE TABLE_NAME IN
   (SELECT TABLE_NAME FROM FND_DF_TABLE_USAGES TBL WHERE
   DESCRIPTIVE_FLEXFIELD_CODE = flexfield-code
   AND TBL.APPLICATION_ID = application-id
   AND TABLE_TYPE = 'BASE' AND FLEXFIELD_USAGE_CODE = flexfield-usage-code)
   ```

To add an extensible flexfield as a single flow for multiple contexts:

1. Open the page to which you want to add the extensible flexfield.

2. From the Data Controls panel, drag the view object for the flexfield-enabled object onto the page or Structure window.

3. When prompted, choose Applications and then choose Oracle Extensible Flexfield.

4. In the User Interface region in the Create Extensible Flexfield dialog, select Panel Splitter or Single Page, as shown in Figure 24-11.
5. Accept the default group code or change its value.

6. If you selected the single page layout, enter the page code for the logical page to be displayed at runtime.

7. Specify the usage code.

8. In the **Object Keys** region, specify the values to use for the category code and the primary keys. The expression/value provided is evaluated at runtime to determine the object instance for which the extensible flexfield attribute values are to be shown.

9. Click **Apply**.

10. (Optional) Complete the following steps to modify the **ReadOnly** and **_eff_param1** through **_eff_param5** properties:
   
   a. Right click in the page and choose **Go to Page Definition**.
      
   b. In **Executables** section in the page definition, select **taskFlow - ExtensibleFlexfieldRuntimeUI1** and click the **Edit** icon.
      
   c. In the **Input Parameters** region in the **Edit Task Flow Binding** dialog, set **ReadOnly** to **Y** if the page should render all segments as read-only fields. Otherwise, set to **N** or leave the parameter value blank.
      
   d. If you need to pass in custom parameters for UI customization, as described in **How to Customize the Runtime User Interface Modeler for Extensible Flexfields**, use the **_eff_param1** through **_eff_param5** parameters to specify the custom parameter values. For example, if you need to customize the page to display or not display certain segments based upon the user’s location, you can use one of these parameters to provide the location as an input parameter.

11. If you have not done so already, complete the following substeps:
   
   a. Open the **Databindings.cpx** file and add the following **EFFRuntimeAM** data control:
b. Add the following code to the Application tag at the top of the
Databindings.cpx file to enable the application to find the page
definitions for the generated UI artifacts:

```xml
PageMapClass="oracle.jbo.uicli.mom.DynamicPageMapImpl"
BaselineDefPackageName="pageDefs"
```

### 24.9.2 How to Expose the Complete Set of an Extensible Flexfield’s Usages, Logical
Pages, and Associated Contexts

To expose the complete set of usages, logical pages, and associated contexts, build a UI
page that includes a splitter with two task flows: one containing the list of extensible
flexfield usages and their associated logical pages on the left, and the other containing
the contexts associated with the selected logical page on the right.

**To build the UI page:**

1. Create a task flow for the extensible flexfield usages and a task flow for the
associated contexts.
2. Create the left and right fragment pages.
3. Use the task flows in the UI page.

#### 24.9.2.1 Creating the Task Flows

The UI page requires two task flows: one containing the list of extensible flexfield
usages and their associated logical pages, and the other containing the contexts
associated with the selected logical page.

**Before you begin:**

Create and configure business components to support the extensible flexfield as
described in *Creating Extensible Flexfield Business Components*.

**To create the two task flows:**

1. Create a page-list task flow, for example, `PageListTF`.
2. Add the following parameter to the task flow:
   - **Name:** `ContainerBean`
   - **Class:**
     ```java
     oracle.apps.fnd.applcore.flex.eff.runtime.EffCategoryPages
     ```
3. Create the context container task flow, for example, ContextPageTF.

4. Add a parameter that is the same as the one you added in Step 2 to the task flow.

24.9.2.2 Creating the Fragments

Create a left fragment for the list of flexfield usages and a right fragment for contexts.

Before you begin:

Create the page-list task flow and context container task flow as described in Creating the Task Flows.

To create the fragments:

1. Create the left and right fragment pages PageListFrag.jsff and ContextPageFrag.jsff.

2. Add the EffCatPageListContainer seeded task flow from the ViewController FlexModeler-View library to the page list fragment using the following parameters:
   - _eff_application_id: Provide the FND_EF_UI_PAGES_B.APPLICATION_ID value; for example 10010.
   - _eff_descriptive_flexfield_code: Provide the FND_EF_UI_PAGES_B.DESCRIPTIVE_FLEXFIELD_CODE value; for example EGO_ITEM_EFF.
   - _eff_category_code: Provide the FND_EF_UI_PAGES_B.CATEGORY_CODE value; for example ELECTRONICS.
   - _eff_usage_code: Provide the FND_EF_UI_PAGES_B.FLEXFIELD_USAGE_CODE value; for example, EGO_ITEM_DL.
   - _eff_containerBean: Provide #{pageFlowScope.ContainerBean}.

3. Add the EffContextsPageContainer seeded task flow from the ViewController FlexModeler-View library to the context page fragment using the following parameters:
   - _eff_application_id: Provide #{pageFlowScope._eff_application_id}.
   - _eff_descriptive_flexfield_code: Provide #{pageFlowScope._eff_descriptive_flexfield_code}.
   - _eff_category_code: Provide #{pageFlowScope._eff_category_code}.
   - _eff_usage_code: Provide #{pageFlowScope._eff_usage_code}.
   - _eff_page_code: Provide #{pageFlowScope._eff_page_code}.
   - _eff_containerBean: Provide #{pageFlowScope.ContainerBean}.
• \_eff\_category\_pk1: (optional) Provide the value for the category’s first primary key column, if applicable. For example, if the first primary key column is INVENTORY\_ITEM\_ID, and its value is 149, provide 149.

• \_eff\_category\_pk2 - (optional) Provide the value for the category’s second primary key column, if applicable. For example, if the second primary key column is ORGANIZATION\_ID, and its value is 204, provide 204.

• \_eff\_category\_pk3 through pk5: Provide the value for that primary key column, if applicable.

• \_eff\_check\_privilege: (optional) Set to Y if the view privilege is to be honored for the context regions.

4. (Optional) To make it easy for implementors to identify which region on a page contains the extensible flexfield, set the Text property for the region component that contains the extensible flexfield task flow to #{applcoreBundle.EFF\_REGION}.

5. Drag and drop the page list fragment onto the page-list task flow.

6. Drag and drop the context page fragment onto the context task flow.

24.9.2.3 Using the Task Flows in the Page

Use the task flows that you just created in Creating the Task Flows to add the usages, logical pages, and associated context lists to the page.

Before you begin:

1. Create the page-list task flow and context container task flow as described in Creating the Task Flows.

2. Create the left and right fragment pages PageListFrag.jsff and ContextPageFrag.jsff as described in Creating the Fragments.

To use the task flows in the page:

1. Open the page and add the splitter.

2. Add the page-list and context page task flows to the left and right facets of the splitter.

3. Enter the value of pageFlowScope.EffCategoryPagesBean for the ContainerBean parameter to the task flows.

4. If you have not done so already, complete the following substeps:
   a. Open the Databindings.cpx file and add the following EFRuntimeAM data control:

      <BC4JDataControl id="EFRuntimeAMDataControl"
      Package="oracle.apps.fnd.applcore.flex.eff.runtime.applicationModule"
      FactoryClass="oracle.adf.model.bc4j.DataControlFactoryImpl"
      SupportsTransactions="true" SupportsFindMode="true"
      SupportsRangesize="true" SupportsResetState="true"
      SupportsSortCollection="true" Configuration=""/>
b. Add the following code to the Application tag at the top of the Databindings.cpx file to enable the application to find the page definitions for the generated UI artifacts:

```xml
PageMapClass="oracle.jbo.uicli.mom.DynamicPageMapImpl"
BasePageDefPackageName="pageDefs"
```

### 24.9.3 How to Expose One Extensible Flexfield Context

Build an application page with a single task flow as a dynamic region on the right, and containing the context that was passed. This variation does not present an extensible flexfield usage or page-list task flow in the user interface.

**Before you begin:**

Create and configure business components to support the extensible flexfield as described in [Creating Extensible Flexfield Business Components](#).

**To expose the extensible flexfield context:**

1. Create a task flow for Pages List, for example, `PageListTF.xml`.
2. Add a managed bean called `EffCategoryPagesBean` to this task flow. It should have a class of `pageFlowScope.EffCategoryPagesBean` and a scope of `pageFlow`.
3. Create a fragment, for example `PageListFrag.jsff`, and drag and drop the `EffCatPageListContainer` task flow from the ViewController FlexModeler-View library on the fragment.
4. Add the following parameters:
   - `_eff_application_id`: Provide `#{pageFlowScope._eff_application_id}`.
   - `_eff_descriptive_flexfield_code`: Provide `#{pageFlowScope._eff_descriptive_flexfield_code}`.
   - `_eff_category_code`: Provide `#{pageFlowScope._eff_category_code}`.
   - `_eff_usage_code`: Provide `#{pageFlowScope._eff_usage_code}`.
   - `_eff_page_code`: Provide `#{pageFlowScope._eff_page_code}`.
   - `_eff_containerBean`: Provide `#{pageFlowScope.EffCategoryPagesBean}`.
   - `_eff_check_privilege`: (optional) Set to `Y` if the view privilege is to be honored for the context regions. [[Check with Gaurav whether this should be included]]
5. Place the task flow on any page on which you want the page list to appear.
6. Create a task flow, for example, `ContextsTF.xml`.
7. Add an empty fragment to this task flow, for example, `ContextsFrag.jsff`.

8. Access the page on which you would like your contexts to appear. This could possibly be the same page as in Step 5, but is likely the second facet of a splitter.

9. Drag and drop `ContextsTF.xml` as a dynamic region onto this page at the correct location.

10. When prompted for a backing bean, enter a new bean name, class, and package, for example, `ContextsRenderingBean.java`, and allow it to remain as a backing bean.

11. Open the Java file and add the following code, replacing the value for `taskFlowId` with the appropriate string:

```java
public class ContextsRenderingBean {
    private String taskFlowId = "/WEB-INF/oracle/apps/fnd/applcore/flex/eff/runtime/ui/test/flow/ContextsTF.xml#ContextsTF";
    private String newTaskFlowId = null;
    private boolean refreshRegion = false;

    public ContextsRenderingBean () {
    }

    public TaskFlowId getDynamicTaskFlowId() {
        return TaskFlowId.parse(taskFlowId);
    }

    public boolean getRefreshRegion() {
        refreshRegion = false;
        newTaskFlowId = (String) ADFContext.getCurrent().getSessionScope().get("_eff_context_tf_id");

        if (newTaskFlowId != null && taskFlowId.compareTo(newTaskFlowId) != 0) {
            taskFlowId = newTaskFlowId;
            refreshRegion = true;
        }
        return refreshRegion;
    }
}
```

12. Open the page definition for the launch page and scroll to the location of the dynamic region binding.

13. Add the following refresh condition setting to the `taskFlow` tag:

```
RefreshCondition="#{backingBeanScope.ContextsRenderingBean.refreshRegion}"
```

This setting provides the callback and the conditions for the refresh of the contexts region.

When you run the page and click the page link, the callback to `refreshRegion` will look up the ID for the contexts task flow and enable a refresh condition of the dynamic region.

14. If you have not done so already, complete the following substeps:

   a. Open the `Databindings.cpx` file and add the following `EFFRuntimeAM` data control:

```
<BC4JDataControl id="EFFRuntimeAMDataControl"
     Package="oracle.apps.fnd.applcore.flex.eff.
```
runtime.applicationModule="
FactoryClass="oracle.adf.model.bc4j.DataControlFactoryImpl"
SupportsTransactions="true" SupportsFindMode="true"
SupportsRangesize="true" SupportsResetState="true"
SupportsSortCollection="true" Configuration="EFFRuntimeAM"
syncMode="Immediate"
xmlns="http://xmlns.oracle.com/adfm/datacontrol"/>

b. Add the following code to the Application tag at the top of the Databindings.cpx file to enable the application to find the page definitions for the generated UI artifacts:

PageMapClass="oracle.jbo.uicli.mom.DynamicPageMapImpl"
BasePageDefPackageName="pageDefs"

24.9.4 How to Expose Extensible Flexfield Attributes in a Table

You can expose extensible flexfield attributes as columns in a table that displays attributes for a specific category. The attributes from all the contexts that are associated with the selected category are displayed. The attributes are grouped in a column span and are labeled context:segment.

Before you begin:

- If you have not done so already, add the stub search view object that was created when you ran the Create Flexfields Business Components wizard to an application module. The stub search view object is in the view folder in the package that you registered for the extensible flexfield usage in the FND_DF_ADFBC_USAGES table.

note:
The Create Flexfields Business Components wizard creates the stub search view object based on the flexfield’s declarative SQL-based view object that was registered for the flexfield usage. If no declarative SQL-based view object was registered, then a stub search view object will not exist. You must create and register the SQL-based view object as described in Creating a Declarative SQL-Based View Object to Enable Searching and then run the Create Flexfield Business Components wizard as described in How to Create Business Components For an Extensible Flexfield’s Usage.

To expose category attributes in a table:

1. Open the page to which you want to add the table with extensible flexfield columns.
2. Drag the stub search view object and drop it on the page.
3. When prompted, select Applications > Table.
4. Rearrange the columns and select table options as required. If you choose the filtering option, the table will be a filtered table.
5. Right-click in the page and choose Go to Page Definition.
6. In the source editor for the page definition, remove the AttrNames tag and its nested tags from the nodeDefinition tag in the tree tag. The tags shown in bold in Example 24-5 are examples of the tags to remove.

7. Write down the value for IterBinding.

8. In the source editor for the search page, add the following <af:column> tag to the <af:table> tag. Change iter-binding to the IterBinding value for the tree tag in the page definition. This is the value that you noted in Step 7.

```xml
<af:column headerText="header-text">
  <fnd:extensibleFlexfieldColumn
    value="#{bindings.iter-binding}" columnSpan="false"
    readOnly="true"/>
</af:column>
```

The columnSpan="false" tag turns off column grouping for the table. The value is set to false by default. Set columnSpan to true if you want to group the columns.

9. Open the source code in which you will call the executeQuery method for the view object.

10. Add a call to the EFFSearchUtil.prepareEFFDeclarativeVOForCategory method followed by a call to the view object's executeQuery method.

    The prepareEFFDeclarativeVOForCategory method prepares the declarative view object with all the attributes that pertain to the specified category.

    Example 24-4 shows the API for the prepareEFFDeclarativeVOForCategory method.

11. After the call to the executeQuery method, call the DescriptiveFlexfield.updateFlexColumns method with the table component object as the argument.

    This method refreshes the table to add the flexfield columns for the category.

Example 24-3  AttrNames Tag to Remove from the Page Definition

```xml
<tree
  IterBinding="j_ExtendedDeclarativePublic1Iterator"
  id="j_ExtendedDeclarativePublic1">
  <nodeDefinition DefName="oracle.apps.fnd.applcore.flex.eff.test.genjbo.base.model.view.
    j_ExtendedDeclarativePublicVO" Name="j_ExtendedDeclarativePublic10">
    <AttrNames>
      <Item Value="Description"/>
      <Item Value="ItemType"/>
      <Item Value="Name"/>
      <Item Value="CategoryCode"/>
    </AttrNames>
  </nodeDefinition>
</tree>
```

Example 24-4  prepareEFFDeclarativeVOForCategory API

```java
@param effDeclarativeViewObject : The view object instance on which the table has been built. This should be an instance of j_ExtendedDeclarative<group>VO

@param categoryCode : The category for which to render the table.
```
public static void prepareEFFDeclarativeVOForCategory(
  ViewObjectImpl effDeclarativeViewObject,
  String categoryCode
)

24.10 Incorporating Extensible Flexfields into a Search Form

You can include extensible flexfield view object attributes as search criteria in a query search form. The search form will include an Add Fields button, which displays a list of the flexfield view object attributes. End users can select which attributes of the flexfield view object to use as search criteria.

24.10.1 How to Incorporate an Extensible Flexfield into a Search Form

You incorporate an extensible flexfield into a query search form by dropping a view criteria component from the Data Controls panel onto the page.

Before you begin:

- If you have not done so already, add the stub search view object that was created when you ran the Create Flexfields Business Components wizard to an application module. The stub search view object is in the view folder in the package that you registered for the extensible flexfield usage in the FND_DF_ADFBC_USAGES table.

  Note:
The Create Flexfields Business Components wizard creates the stub search view object based on the flexfield's declarative SQL-based view object that was registered for the flexfield usage. If no declarative SQL-based view object was registered, then a stub search view object will not exist. You must create and register the SQL-based view object as described in Creating a Declarative SQL-Based View Object to Enable Searching and then run the Create Flexfield Business Components wizard as described in How to Create Business Components For an Extensible Flexfield's Usage.

- If the desired view criteria for the stub search view object does not exist, create it. The view criteria must contain the flexfield’s CategoryCode attribute. The operator for the CategoryCode attribute must be set to Equals.

To incorporate an extensible flexfield into a search form:

1. Open the search page to which you want to add the extensible flexfield.

2. From the Data Controls panel, expand the stub search view object, expand Named Criteria, and drag the desired view criteria onto the page or Structure window.

3. Use the stub search view object to create a results panel and a query panel as described in the Creating Query Search Forms section in the section in the Developing Fusion Web Applications with Oracle Application Development Framework.

4. Right-click in the page and choose Go to Page Definition.

5. In the source editor for the page definition, remove the AttrNames tag and its nested tags from the nodeDefinition tag in the tree tag. The tags shown in bold in Example 24-5 are examples of the tags to remove.
6. Write down the value for IterBinding.

7. In the Structure window for the search page, select the af:query component.

8. In the Property Inspector, expand Behavior and set the QueryListener property to FlexEFFSearchUtilBean.effQueryListener.

9. In the source editor for the search page, add the following <af:column> tag to the <af:table> tag. Change iter-binding to the IterBinding value for the tree tag in the page definition. This is the value that you noted in Step 6.

   <af:column headerText="header-text">
     <fnd:extensibleFlexfieldColumn
       value="#{bindings.iter-binding}"
       readOnly="true"/>
   </af:column>

Example 24-5  AttrNames Tag to Remove from the Page Definition

   <tree
      IterBinding="j_ExtendedDeclarativePublic1Iterator"
      id="j_ExtendedDeclarativePublic1">
     <nodeDefinition
       DefName="oracle.apps.fnd.applcore.flex.eff.test.genjbo.base.model.view.j_ExtendedDeclarativePublicVO"
       Name="j_ExtendedDeclarativePublic10">
       <AttrNames>
         <Item Value="Description"/>
         <Item Value="ItemType"/>
         <Item Value="Name"/>
         <Item Value="CategoryCode"/>
       </AttrNames>
     </nodeDefinition>
   </tree>

24.11 Preparing Extensible Flexfield Business Components for Oracle Business Intelligence

Oracle Business Intelligence is a comprehensive collection of enterprise business intelligence functionality that provides the full range of business intelligence capabilities including interactive dashboards, full ad hoc, proactive intelligence and alerts, enterprise and financial reporting, real-time predictive intelligence, and more.

Extensible flexfields are modeled using polymorphic view objects, which are not compatible with Oracle Business Intelligence. To enable an extensible flexfield to be used by Oracle Business Intelligence, the corresponding polymorphic view objects must be flattened into a usable static form.

If an extensible flexfield is business-intelligence enabled and a declarative SQL-based view object is registered for the flexfield, then, when you create business components for the extensible flexfield, the business component modeler generates a stub view object for business intelligence alongside the standard extensible flexfield polymorphic view object.

At runtime, the implementor can select which individual flexfield segments to make available for use with Oracle Business Intelligence. Those segments are included in a flattened view object that is generated from the stub view object. Administrators must then import the flexfield into the Oracle Business Intelligence repository to make it available for Oracle Business Intelligence.
24.11.1 How to Enable an Extensible Flexfield for Oracle Business Intelligence

To cause the business component modeler to generate the stub view object for business intelligence, you must business-intelligence enable the flexfield. Use the setup APIs to set the flexfield's BI_ENABLED flag business-intelligence as described in Using the Setup APIs to Register and Define Extensible Flexfields.

You can optionally use the setup APIs to provide flattened fact names for the flexfield’s entity details. You provide a flattened fact name by setting the BI_FLATTENED_FACT_NAME value.

24.11.2 How to Flatten the Extensible Flexfield Model for a Business Intelligence-Enabled Extensible Flexfield

When you create business components for a business intelligence-enabled extensible flexfield, the business component modeler recognizes the business intelligence-enabled setting. A view object that is flattened for Oracle Business Intelligence is generated alongside the standard extensible flexfield polymorphic view object.

Before you begin:

1. Ensure that a business intelligence application module and a business intelligence view object exist for the product table for which you are implementing the extensible flexfield. For information, see Designing and Securing View Objects for.

2. Create a declarative SQL-Based view object as described in Creating a Declarative SQL-Based View Object to Enable Searching.

3. Ensure that the declarative SQL-Based view object was registered for the flexfield, either by using the Register Flexfield Metadata wizard to provide the fully qualified name of the search view object or by executing the following SQL statement to register it. Replace search-view-object with the fully qualified name for the search view object. Replace product-table with the name of the product table that the flexfield extends.

   ```sql
   UPDATE FND_DF_ADFBC_USAGES
   SET EFF_SEARCH_VIEW_OBJECT = 'search-view-object'
   WHERE TABLE_NAME='product-table'
   AND ENTITY_OBJECT='product-table-entity-object';
   
   4. Enable the flexfield and the desired segments for Oracle Business Intelligence as described in How to Enable an Extensible Flexfield for.

To produce a business intelligence-enabled flattened extensible flexfield model:

1. Create extensible flexfield business components as described in How to Create Business Components For an Extensible Flexfield’s Usage.

   For a flexfield that is business intelligence-enabled, the Create Flexfield Business Components wizard generates a stub view object, which is named object-name-prefixEFFBFIFlattenedVO, and other business components under a directory called analytics in the package root directory. These are generated in addition to the standard extensible flexfield view object.

2. Complete the following steps to link the stub view object to the business intelligence view object for the product table.
a. From the File menu, choose New.

b. In the New Gallery, go to Business Tier > ADF Business Components and select Flexfield View Link.

c. Click OK to access the Create Flexfield View Link wizard.

 d. On the Name page, from the Package dropdown list, specify a package for the view link.

Caution:
You cannot move the view link to a different package after you create it. Instead, you must delete the view link and re-create it with the new package name.

 e. In the Name field, enter a name for the view link.

 f. Click Next.

 g. In the Select Source View Object tree, expand the available objects from your current project and select the master view object, which is the business intelligence view object for the product table.

h. In the Select Destination Flexfield tree, expand the application and select the destination flexfield usage, which is the stub view object.

i. In the View Link Accessor Name field, enter an appropriate name for the view link accessor.

j. Click Next.

k. On the Source Attributes page, click Finish.

For extensible flexfields, the Source Attributes page is informational only. The wizard uses the primary key attributes of the source view object to define the view link.

Note:
You can skip the Properties page because view link-specific properties are not supported.

l. On the Summary page, click Finish.

3. (Optional) To control how the extensible flexfield attributes appear in the Oracle Business Intelligence presentation layer, edit the application module that contains the stub view, and then, in the Custom Properties section in the General navigation tab, add one or more of the following properties:

- To display only the attributes in the stub view object that have a specific context usage or category code, add one of the following custom properties and values to the application module:
  
  - FND_ACFF_SHOW_ATTRIBUTES_WITH_PROPERTY_stub-vo-instance-name = "FND_ACFF_EFF_CONTEXT_USAGE=propertyvalue"
For example, the following custom property specifies that when displaying the attributes from the ItemEFFBIFlattenedVO view object, only the attributes that have the FND_ACFF_EFF_CONTEXT_USAGE property with a value of EGO_ITEM_DL will be displayed in the presentation layer:

\[
\text{FND_ACFF_SHOW_ATTRIBUTES_WITH_PROPERTY}\_\text{stub-vo-instance-name} = "\text{FND_ACFF_EFF_CATEGORY_CODE=propertyvalue}\"
\]

**Note:**

If the property is not set, then all attributes appear in the presentation layer.

- To specify how to arrange the attributes in the presentation layer, add one of the following properties:
  - To arrange the attributes by context within category, add the following custom property:
    \[
    \text{FND_ACFF_EFF_BI_PRESENTATION_OPTION}\_\text{stub-vo-instance-name} = "\text{FLAT_CATEGORY_CONTEXT_FOLDER_LIST}\"
    \]
    The attributes will be displayed in context folders within each applicable category folder.
  - To arrange the attributes by category within context, add the following custom property:
    \[
    \text{FND_ACFF_EFF_BI_PRESENTATION_OPTION}\_\text{stub-vo-instance-name} = "\text{FLAT_CATEGORY_FOLDER_LIST_NESTED_CONTEXT}\"
    \]
    The attributes will be displayed in category folders within each applicable context folder. This is the default option.

**Note:**

If the FND_ACFF_EFF_BI_PRESENTATION_OPTION\_stub-vo-instance-name option is not defined, then the attributes are arranged by category within context.
Business Intelligence Publisher provides a mechanism to extract data from a remote server using web service calls to a View Object. Applications developers can define data sources as View Objects in their application and then create a data model in BI Publisher to retrieve the data to use in their reports. Application developers must register an AM with BI Publisher which includes the View Objects which will fetch data for the reports. To allow users to use Extensible Flexfield segment attributes to define reports in BI Publisher, these flexfield segments need to view link the stub for a flattened View Object generated by Flex Modelers. The flattened View Object will include all segments defined in the flexfield across contexts and categories.

24.13 Loading Seed Data

Any implementation of flexfields in Oracle Fusion Applications typically requires application seed data, which is the essential data that enables flexfields to work properly in applications. Flexfield seed data can be uploaded and extracted using Seed Data Loader.

After you complete the registration process described in How to Register Extensible Flexfields, your flexfield seed data consists of the information that you registered for your flexfield, such as the tables and columns reserved for your flexfield. For a customer flexfield, the seed data contains only this registration data.

If your flexfield is a developer flexfield, you also serve as the implementor. In addition to the registration data, your flexfield seed data might include contexts, segments, and value sets that you have defined for your flexfield. You can use the Oracle Fusion Middleware Extensions for Applications (Applications Core) Setup application, as described in Deploying Flexfields in a Standalone WebLogic Server Environment, to configure a flexfield.

Use the Manage Extensible Flexfields and Manage Value Sets tasks or use the PL/SQL APIs to create contexts, segments, and value sets, and then use the seed data loader to extract your seed data. To protect flexfield seed data from customer edits, use the Manage Extensible Flexfields task to set flags to mark flexfield pages and contexts as protected. Use the Manage Value Sets task to set the flag to protect a value set.

Only the user with the SEED_DATA_FROM_APPLICATION_USER user name can modify protected flexfield pages, contexts, and value sets from the setup tasks, such as the Manage Extensible Flexfields and Manage Value Sets tasks.

For information about extracting and loading seed data, see Initializing Oracle Fusion Application Data Using the Seed Data Loader.

24.14 Customize the Extensible Flexfield Modelers

You can customize the modelers that generate the extensible flexfield business component and UI artifacts to add additional product-specific logic. For example, you might want the modeler to set a specific JavaServer Pages (JSP) Expression Language (EL) expression on a generated segment's ReadOnly property.

24.14.1 How to Customize the Runtime Business Component Modeler for Extensible Flexfields

To extend the runtime business component modeler for extensible flexfields, override the EFFBCModelerFactory.getCustomEFFBCModeler method, shown in Example 24-6, in the custom runtime business component modeler factory for extensible flexfields.
An example of the PIMBCModelerFactory.java override for this method is shown in Example 24-7.

Example 24-6  getCustomEFFBCModeler Method

```java
/**
 * Implementing teams need to override this method and provide the custom BC Modeler here (for example, PIMBCModeler).
 * @param namespace the namespace
 * @param flexDef the flexfield def
 * @param entityUsage the entity usage
 * @param writer the modeler writer
 * @param conf the configuration
 * @param categoryDef the category def
 * @param categoryContextDef the category context def
 * @param categoryCode the category code
 * @param contextCode the context code
 * @param appShortName the application short name
 * @param flexCode the flexfield code
 * @param connUrl the connection url
 * @param isInterface the interface flag
 * @return the custom EFF BC Modeler instance.
 */
protected EFFBCModeler getCustomEFFBCModeler(FlexfieldNamespace namespace,
                                              FlexfieldDef flexDef,
                                              FlexfieldEntityUsage entityUsage,
                                              BCModelerWriter writer,
                                              Map<BCModeler.Option, Object> conf,
                                              CategoryDef categoryDef,
                                              CategoryContextDef categoryContextDef,
                                              String categoryCode,
                                              String contextCode,
                                              String appShortName,
                                              String flexCode,
                                              String connUrl,
                                              boolean isInterface)
{
    return new EFFBCModeler(namespace, flexDef, entityUsage, writer, conf,
                              categoryDef, categoryContextDef, categoryCode,
                              contextCode, appShortName, flexCode, connUrl,
                              isInterface);
}
```

Example 24-7  PIMBCModelerFactory.java Override

```java
protected EFFBCModeler getCustomEFFBCModeler(FlexfieldNamespace namespace,
                                              FlexfieldDef flexDef,
                                              FlexfieldEntityUsage entityUsage,
                                              BCModelerWriter writer,
                                              Map<BCModeler.Option, Object> conf,
                                              CategoryDef categoryDef,
                                              CategoryContextDef categoryContextDef,
                                              String categoryCode,
                                              String contextCode,
                                              String appShortName,
                                              String flexCode,
                                              String connUrl,
                                              boolean isInterface)
{
    return new PIMBCModeler(namespace, flexDef, entityUsage, writer, conf,
                              categoryDef, categoryContextDef, categoryCode,
                              contextCode, appShortName, flexCode, connUrl,
```

Customize the Extensible Flexfield Modelers
24.14.2 How to Customize the Runtime User Interface Modeler for Extensible Flexfields

If the UI artifacts that are generated for an extensible flexfield business component do not fulfill application requirements, you can create wrapper implementation classes for the framework's customizer interfaces. These wrapper implementation classes enable some control over the XML code that the modeler generates for the UI artifacts just before it persists the generated task flows and JavaServer Faces (JSF) page fragments. The implementation classes are in the oracle.apps.fnd.applcore.flex.uimodeler.customizers package.

To customize an extensible flexfield business component's generated UI artifacts:

1. Create wrapper classes for the default customizer implementation classes.
2. Create a wrapper of the metadata provider implementation class.
3. Register the metadata provider wrapper class in the metadata for the flexfield's business component.

24.14.2.1 Create the Customizer Wrapper Class

Extend the default customizer implementation classes from the oracle.apps.fnd.applcore.flex.uimodeler.customizers package to customize the following UI artifacts:

- Context JSF fragment
- Segment components in the generated context task flow
- Page links in the generated links task flow
- Page task flow
- Search task flow

24.14.2.1.1 How to Customize the Context JSF Page Fragment

To customize the JSF page fragment for a single row context, create a wrapper class for the SingleRowContextRegionCustomizerImpl implementation class and override the customizeGeneratedSingleRowContextFragment method.

To customize the JSF page fragment for a multiple row context, create a wrapper class for the MultiRowContextRegionCustomizerImpl implementation class and override the customizeGeneratedMultiRowContextFragment method.

24.14.2.1.2 How to Customize the Segment Components in the Generated Context Task Flow

To customize which segment components in the generated context task flow will be read-only for single and multirow contexts, create a wrapper class for the ContextComponentsCustomizerImpl implementation class and override the getAttributeComponentReadonlyELExpression method. This method returns the value of the readonly property for a given segment component in the context task flow, as shown in Example 24-8.

Example 24-8  Sample getAttributeComponentReadonlyELExpression Method

```java
public String getAttributeComponentReadonlyELExpression(
    Transaction tx, ViewDef contextViewDef) {
```
24.14.2.1.3 How to Customize the Page Links in the Generated Links Task Flow

To customize which page links in a generated links task flow will be rendered, create a wrapper class for the PageListCustomizerImpl implementation class and override the getPageLinkRenderedProperty method. This method returns the value of the rendered property for a page link.

24.14.2.1.4 How to Customize the Page Task Flow

To customize which context task flows will be rendered in a page task flow, create a wrapper class for the ContainerPageCustomizerImpl implementation class and override the getContextTFRenderedELExpression method. This method returns the value of the region tag for a context task flow.

If implementors have defined a view privilege on a context, then the application checks if the context is present in either the base or translation extension table before verifying whether the end user has view privileges for the context. By default, the application uses the primary keys of the product table plus the context code and category code to search for a matching row in the extension tables. If your application uses a different method for associating extension table rows with product table rows, you must override the prepareSqlQueryForMatchingEffLineId method in the oracle.apps.fnd.appcore.flex.eff.runtime.EffCategoryPagesBean.java class to return the search query that is to be used to find a matching row. This method takes the security object, the context code, the category code, and the category row as input parameters and it returns the query in String form.

24.14.2.1.5 How to Customize the Search Task Flow

To customize the search task flow, create a wrapper class for the SearchRegionCustomizerImpl implementation class. Table 24-3 shows the ways in which you can customize the search task flow and the methods to override to perform the customizations.

<table>
<thead>
<tr>
<th>Customization</th>
<th>Method to Override</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set the list of actions required in the search results table.</td>
<td>getResultTableActionMenu</td>
<td>Use the ResultsTableActionMenu inner class to define the menu properties and entries.</td>
</tr>
<tr>
<td>Set the managed bean for the generated task flow.</td>
<td>getSearchManagedBeanClass</td>
<td>The manage bean must extend oracle.apps.fnd.appcore.flex.eff.search.ui.bean.DefaultEffSearchManagedBean.</td>
</tr>
<tr>
<td>Customization</td>
<td>Method to Override</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Alter properties in the product table component for search results.</td>
<td>getApplicationsTablePropertiesMap</td>
<td>This method returns a hashmap of name-value pairs of the properties to be set.</td>
</tr>
<tr>
<td>Alter properties in the ADF Table that is in the product table component for search results.</td>
<td>getADFTablePropertiesMap</td>
<td>This method returns a hashmap of name-value pairs of the properties to be set, as shown in Example 24-9.</td>
</tr>
<tr>
<td>Customize the query panel properties.</td>
<td>getQueryPanelPropertiesMap</td>
<td>This method returns a hashmap of name-value pairs of the properties.</td>
</tr>
<tr>
<td>Get a handle to the document object model (DOM) object of the generated search region and its pageDef.</td>
<td>customizeGeneratedSearchTaskflowRegion</td>
<td>Use DOM objects to customize generated artifacts only if there is no other way to perform for the customization.</td>
</tr>
<tr>
<td>Customize the readOnly property for a component in search results table column.</td>
<td>getResultsTableColumnComponentReadOnlyELExpression</td>
<td>This method returns a JSP EL expression.</td>
</tr>
<tr>
<td>Customize the properties on the search results table column for an attributeDef.</td>
<td>getResultsTableColumnPropertiesMap</td>
<td>This method returns a hashmap of name-value pairs of the properties.</td>
</tr>
<tr>
<td>Set the task flow data control scope to shared.</td>
<td>getSearchTaskFlowDataControlScope</td>
<td>The task flow can be generated with either a SHARED scope or an ISOLATED scope. The default is ISOLATED. Override this method to set the scope to SHARED.</td>
</tr>
<tr>
<td>Add custom toolbar components.</td>
<td>getAdditionalToolbarComponentDefinitions</td>
<td>Use the AdditionalToolbarComponentDefinition inner class to provide metadata for the required component. You can add components of type CHOICELIST, BUTTON, SPACER, and SEPARATOR.</td>
</tr>
</tbody>
</table>
Table 24-3  (Cont.) Methods to Override to Customize the Search Task Flow

<table>
<thead>
<tr>
<th>Customization</th>
<th>Method to Override</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set the default search criteria on the search page as it is loaded.</td>
<td>getDefaultSearchCriteriaName</td>
<td>NA.</td>
</tr>
<tr>
<td>Customize the generated search task flow XML.</td>
<td>customizeGeneratedSearchTaskflow</td>
<td>This method returns a handle to the DOM object for the generated task flow XML. You should use DOM objects to customize generated artifacts only if there is no other way to perform the customization.</td>
</tr>
</tbody>
</table>

Example 24-9  Sample getADFTablePropertiesMap Method

```java
public HashMap getADFTablePropertiesMap()
{
    HashMap propertiesMap = new HashMap(10);
    propertiesMap.put("filterVisible","false");
    propertiesMap.put("columnStretching","column:description");
    propertiesMap.put("inlineStyle","width:100%;");
    return propertiesMap;
}
```

24.14.2.1.6 How to Create a Metadata Provider Implementation Class

If you have created customizer wrapper classes for a flexfield business component, you must create a wrapper of the default UIModelerMetadataProviderImpl implementation class for that business component. This class is in the oracle.apps.fnd.applcore.flex.uimodeler package.

Override the appropriate methods in the following list to return the names of the custom wrapper classes. For example, if you created a custom wrapper class to customize the search task flow, you would override the getSearchRegionCustomizerClassName method, as shown in Example 24-10. Override only the methods that correspond to the classes for which you have created custom wrappers. In the case of a SearchRegionCustomizerImpl custom wrapper, the method that you override depends on whether the wrapper customizes the interface search UI or the regular search UI. If the wrapper customizes the interface search UI, override the getInterfaceSearchRegionCustomizer method. Otherwise, override the getSearchRegionCustomizerClassName method.

- getApplicationContextCustomizerClassName
- getContainerPageCustomizerClassName
- getPageListCustomizerClassName
- getSearchRegionCustomizerClassName
- getInterfaceSearchRegionCustomizer
- getSingleRowContextRegionCustomizerClassName
getMultiRowContextRegionCustomizerClassName

Example 24-10  Sample getSearchRegionCustomizerClassName Method

```java
public String getSearchRegionCustomizerClassName() {
    return "oracle.apps.myProduct.myApp.items.eff.ItemSearchRegionCustomizer";
}
```

24.14.2.1.7 How to Register the Metadata Provider Class for the Business Component

For the user interface modeler to use your custom wrapper classes at runtime, you must register the metadata provider class. To register the class, set the ADFUI_MODELER column for the business component’s row in the FND_DF_FLEXFIELDS_B table to the name of the UIMetadataProviderImpl implementation class that you created for the business component.

24.15 Testing Flexfields

You can test specified flexfields from JDeveloper using the Run with Flexfields dialog or the Debug with Flexfields dialog. You can also test the flexfields by running the application from an Integrated WebLogic Server instance. For more information, see Testing Flexfields.

24.16 Making Extensible Flexfields Available to Web Services

You can make extensible flexfields available to web service clients and enable the clients to perform create, read, update, and delete (CRUD) operations on the flexfield data rows.

24.16.1 How to Make an Extensible Flexfield Available to Web Services

To make web service access available for an extensible flexfield, associate the flexfield usage’s stub search view object with the service-backed view object for the product table that the extensible flexfield extends. Then add a SERVICE_PROCESS_CHILDREN property to the view link and create service data object (SDO) classes for the service-backed view object.

The stub search view object serves as a placeholder for resolving build time dependencies. When the flexfield is deployed, the flexfield modelers generate a view object with the same name in the MDS repository, and generates the view object’s XML schema definition (XSD). Because of the name space configuration, the generated view object and XSD are picked up instead of the view object that is shipped with the application. The XSD that is generated for the stub search view object includes references to the XSD definitions for the child categories and contexts that the implementor defines at runtime. At runtime, the application uses view object polymorphism to obtain the category and context view objects that extend from the stub search view object.

Before you begin:

- Ensure that a stub search view object exists for the flexfield usage. The stub search view object is in the view folder in the package that you registered for the extensible flexfield usage in the FND_DF_ADFBC_USAGES table. The name of the stub search view object begins with `j_` and ends with the name of the usage’s view object. You can use the following SQL statement to determine the package name and view object name for the flexfield usage.

```sql
SELECT PACKAGE_NAME, VIEW_OBJECT
FROM FND_DF_ADFBC_USAGES
```
WHERE TABLE_NAME IN
(SELECT TABLE_NAME FROM FND_DF_TABLE_USAGES TBL WHERE
DESCRIPTIVE_FLEXFIELD_CODE = flexfield-code AND
TBL.APPLICATION_ID = application-id AND
TABLE_TYPE = 'BASE' AND
FLEXFIELD_USAGE_CODE = flexfield-usage-code)

Note:
The Create Flexfields Business Components wizard creates the stub search view object based on the flexfield's declarative SQL-based view object that was registered for the flexfield usage. If no declarative SQL-based view object was registered, then a stub search view object will not exist. You must create and register the SQL-based view object as described in Creating a Declarative SQL-Based View Object to Enable Searching and then run the Create Flexfield Business Components wizard as described in How to Create Business Components For an Extensible Flexfield’s Usage.

- A service-backed view object must exist for the product table that the flexfield extends, as described in the Creating View Objects Backed by SDO Services section in the Developing Fusion Web Applications with Oracle Application Development Framework.

**To expose an extensible flexfield as a web service:**

1. Create a view link as described in the How to Create a Master-Detail Hierarchy for Entity-Based View Objects section in the Developing Fusion Web Applications with Oracle Application Development Framework.

   a. On the View Objects page, in the Select Source Attribute tree expand the service-based view object in the desired package. In the Select Destination Attribute tree expand the stub search view object.
      
      For entity-based view objects, notice that in addition to the view object attributes, relevant associations also appear in the list.

   b. Select the primary key and CategoryCode attributes in both Source and Destination trees. Then click Add to add the association to the table that is below the trees.

   c. Click Finish.

2. In the Application Navigator, double-click the view link that you just created.

3. In the overview editor, expand the Custom Properties section and add a SERVICE_PROCESS_CHILDREN property set to true, if one does not already exist.

4. Open the service-based view object.

5. In the overview editor click the Java navigation tab.

6. In the Java Classes section, click the Edit icon to generate and configure Java implementation classes.

7. In the Select Java Options dialog, select Generate Service Data Object Class and click OK.
24.17 Access Information About Extensible Flexfield Business Components

The consumers of an extensible flexfield might need to programmatically access an extensible flexfield, for instance, to further process the data that has been entered for an extensible flexfield, to add additional validation, or to perform change control. The oracle.apps.fnd.applcore.flex.runtime.util.common.ExtensibleFlexfieldUtil package provides methods for obtaining the handles to the artifacts that are generated in a customer's instance.

24.17.1 How to Access Information About Extensible Flexfield Business Components

Use the methods in oracle.apps.fnd.applcore.flex.runtime.util.common.ExtensibleFlexfieldUtil to get the names of the following Java business objects (JBOs) that are generated for an extensible flexfield:

- Context entity object: Use `getContextEoName`, which is shown in Example 24-11.
- Context view object: Use `getContextVoName`, which is shown in Example 24-12.
- Context entity association between base and extension entity objects: Use `getCategoryContextAssocName`, which is shown in Example 24-13.
- Context entity/view object attribute for a given segment code: Use `getContextAttributeName`, which is shown in Example 24-14.
- Categories: Use the various methods shown in Example 24-15.
- Search view object attributes: Use `getSearchVoAttributeNames`, which is shown in Example 24-16.

**Example 24-11 Method to Get EFF Context Entity Object Name**

```java
/**
 * @param appId - application id
 * @param flexCode - flexfield code (e.g. EGO_ITEM_UA)
 * @param flexUsageCode - flexfield usage code (e.g. EGO_ITEM_DL)
 * @param txn - DB transaction
 * @param contextCode - context code (e.g. Voltage)
 * @param tableType - table type (e.g. EXTENSION)
 * @param effGroup - eff grouping entity name (public / private)
 * @return
 */
public static String getContextEoName(Long appId, String flexCode,
                                       String flexUsageCode,
                                       DBTransaction txn,
                                       String contextCode,
                                       String tableType,
                                       String effGroup)
```

**Example 24-12 Method to Get EFF Context View Object Name**

```java
/**
 * @param appId - application id
 * @param flexCode - flexfield code (e.g. EGO_ITEM_UA)
 * @param flexUsageCode - flexfield usage code (e.g. EGO_ITEM_DL)
 */
```
Example 24-13 Method to Get Name for EFF Context Entity Association Between Base and Extension Entity Objects

/**
 * @param appId - application id
 * @param flexCode - flexfield code (e.g. EGO_ITEM_UDA)
 * @param flexUsageCode - flexfield usage code (e.g. EGO_ITEM_DL)
 * @param txn - DB transaction
 * @param contextCode - context code (e.g Voltage)
 * @param tableType - table type (e.g. EXTENSION)
 * @param effGroup - eff grouping entity name (public / private)
 * @return
 */
public static String getCategoryContextAssocName(Long appId, String flexCode,
        String flexUsageCode,
        DBTransaction txn,
        String contextCode,
        String tableType,
        String effGroup)

Example 24-14 Method to Get Attribute Name for EFF Context Entity/View Object Given Segment Code (FND_DF_SEGMENTS_VL(SEGMENT_CODE))

/**
 * Get the attribute name for the EFF context view / entity object given the segment code.
 * @param segmentCode - segment code for attribute name.
 * @return attribute name
 */
public static String getContextAttributeName(String segmentCode)

Example 24-15 EFF Category Methods

public static String getCategoryAmNameForWebServices()
public static String getCategoryAmNameForDataEntry()
public static String getCategoryAmNameForInterfaceGeneric()
public static String getCategoryAmNameForInterfaceCategory()
public static String getCategoryAmNameForSearchGeneric()
public static String getCategoryAmNameForSearchCategory()
public static String getCategoryVoNameForDataEntry()
public static String getCategoryVoNameForWebServices()
public static String getCategoryVoNameForInterfaceGeneric()
public static String getCategoryVoNameForInterfaceCategory()
public static String getCategoryContextViewLinkNameForInterfaceGeneric()
public static String getCategoryContextViewLinkNameForInterfaceCategory()
public static String getCategoryContextViewLinkNameForSearchCategory()
public static String getCategoryContextViewLinkNameForSearchGeneric()
public static String getCategoryContextViewLinkNameForWebService()
public static void useServiceProvider()
public static String getCategoryVoNameForSearchCategory()
public static String getCategoryVoNameForSearchGeneric()

Example 24-16  Method to Get Hash Map of View Object Attribute Names for EFF Search View Object
/**
 * Get a map of segment codes to search view object attribute names
 * @param flexUsageCode - flexfield usage code (e.g. EGO_ITEM_DL)
 * @param contextCode - context code (e.g Voltage)
 * @param segmentCodeList - segment codes for the map.
 * @return map of attribute names with key of segment codes
 */
public static HashMap<String, String> getSearchVoAttributeNames(
    String flexUsageCode,
    String contextCode,
    ArrayList<String> segmentCodeList)

24.18 Use the Advanced Features of Extensible Flexfields

Extensible flexfields support the use of interface tables for loading extensible flexfield data. In addition, extensible flexfields support parallel sets of extensible flexfield artifacts.

24.18.1 How to Support Interface Loading of Extensible Flexfield Data

If a consumer of the flexfield wants to support interface loading of extensible flexfield data, create interface tables that correspond to the product (primary) table and the base extension table. If the flexfield has a translation extension table, also create a corresponding interface table for the translation extension table, and create a view for that interface table using a definition statement similar to that shown in Example 24-1. Create entity objects and view objects for these tables and view. Then add entries to the FND_DF_TABLE_USAGES and FND_DF_ADFBC_USAGES tables as shown in Table 24-4 and Table 24-5.

When the flexfield is deployed, a parallel set of ADF Business Components artifacts are created. You can use these artifacts to insert data into the interface tables.

Table 24-4  FND_DF_TABLE_USAGES Values for Flexfield Interface Tables

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTERPRISE_ID</td>
<td>The enterprise ID.</td>
</tr>
<tr>
<td>TABLE_NAME</td>
<td>The name of the interface table.</td>
</tr>
<tr>
<td>TABLE_USAGE_CODE</td>
<td>The code for the flexfield usage.</td>
</tr>
<tr>
<td>APPLICATION_ID</td>
<td>The application ID.</td>
</tr>
<tr>
<td>DESCRIPTIVE_FLEXFIELD_D_CODE</td>
<td>The code for the extensible flexfield.</td>
</tr>
<tr>
<td>FLEXFIELD_USAGE_CODE</td>
<td>The code for the flexfield usage (the save value as the table usage code).</td>
</tr>
</tbody>
</table>
Table 24-4  (Cont.) FND_DF_TABLE_USAGES Values for Flexfield Interface Tables

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_TYPE</td>
<td>For a table that corresponds to the product table, use &quot;BASE_INTERFACE&quot;. For a table that corresponds to the base extension table, use &quot;EXTENSION_INTERFACE&quot;. For a table that corresponds to the translation extension table, use EXTENSION_INTERFACE_TL&quot;.</td>
</tr>
<tr>
<td>COLUMN_NAME</td>
<td>Null</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>Description of the interface table.</td>
</tr>
</tbody>
</table>

Table 24-5  FND_DF_ADFBC_USAGES Values for Flexfield Interface Tables

<table>
<thead>
<tr>
<th>Column</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TABLE_NAME</td>
<td>Name of the interface table.</td>
</tr>
<tr>
<td>TABLE_USAGE_CODE</td>
<td>Flexfield usage code.</td>
</tr>
<tr>
<td>ENTITY_OBJECT</td>
<td>Name of the entity object for the interface table. When adding an entry for an interface table that corresponds to the translation extension table, provide the name the entity object for the translation view.</td>
</tr>
<tr>
<td>EFF_GROUP_NAME</td>
<td>&quot;Private&quot;</td>
</tr>
<tr>
<td>PACKAGE_NAME</td>
<td>Name of the package in which to put the generated interface artifacts. For example oracle.apps.flex.fnd.myFlex.</td>
</tr>
<tr>
<td>OBJECT_NAME_PREFIX</td>
<td>A short unique name. This prefix is used to derive the names of objects that are generated for the usage.</td>
</tr>
<tr>
<td>DEPLOYMENT_STATUS</td>
<td>Null</td>
</tr>
<tr>
<td>DEPLOYMENT_ERROR_MESSAGE</td>
<td>Null</td>
</tr>
<tr>
<td>APPLICATION_MODULE</td>
<td>The application module that contains the view object for this table.</td>
</tr>
<tr>
<td>VIEW_OBJECT</td>
<td>The view object for this table.</td>
</tr>
<tr>
<td>ENTITY_OBJECT_TL</td>
<td>When adding an entry for an interface table that corresponds to the translation extension table, provide the name of the entity object for the interface table that corresponds to the translation extension table.</td>
</tr>
<tr>
<td>EFF_SEARCH_VIEW_OBJECT</td>
<td>When adding an entry for an interface table that corresponds to the base extension table, provide the name of the declarative SQL-based view object that was created to support searching, if one exists.</td>
</tr>
</tbody>
</table>
24.18.2 How to Support Parallel Sets of Extensible Flexfield Artifacts for Different Uses

If you require parallel sets of extensible flexfield artifacts generated for different uses (for example, public view objects and private view objects), replicate the entries in the FND_DF_ADFBC_USAGES table for that flexfield and use a different group name. When the business components are generated for the flexfield, a separate set of components is generated for each group name.

The following figure shows example entries for a flexfield usage. The FND_DF_TABLE_USAGES table contains rows for the product table (USGA_BASE), the usage base extension table (USGA_EFF_B), and the usage translation extension table (USGA_EFF_TL). The FND_DF_ADFBC-USAGES table contains two entries for each entry in the FND_DF_TABLE_USAGES table — one for the Private group and one for the Public group. Each row names the entity object and the view object. The rows for the translation extension table also name the entity object from the translation extension table. Note that not all columns are shown.

**Note:**
In this example, the Public group is necessary because the product that is using the flexfield usage requires a parallel set of extensible flexfield artifacts generated for public view objects.

**Figure 24-12 Example Entity Detail Entries**

<table>
<thead>
<tr>
<th>FND_DF_TABLE_USAGES</th>
<th>FLEXFIELD_USAGE_CODE</th>
<th>TABLE_NAME</th>
<th>TABLE_TYPE</th>
<th>TABLE_USAGE_CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC_EFF</td>
<td>ABC_USG_A</td>
<td>USGA_BASE</td>
<td>BASE</td>
<td>ABC_USG_A</td>
</tr>
<tr>
<td>ABC_EFF</td>
<td>ABC_USG_A</td>
<td>USGA_EFF_B</td>
<td>EXTENSION</td>
<td>ABC_USG_A</td>
</tr>
<tr>
<td>ABC_EFF</td>
<td>ABC_USG_A</td>
<td>USGA_EFF_TL</td>
<td>EXTENSION_TL</td>
<td>ABC_USG_A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FND_DF_ADFBC-USAGES</th>
<th>EFF_GROUP_NAME</th>
<th>TABLE_USAGE_CODE</th>
<th>TABLE_NAME</th>
<th>ENTITY_OBJECT</th>
<th>ENTITY_OBJECT_TL</th>
<th>VIEW_OBJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private</td>
<td>ABC_USG_A</td>
<td>USGA_BASE</td>
<td>BaseEO</td>
<td>BaseVO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Private</td>
<td>ABC_USG_A</td>
<td>USGA_EFF_B</td>
<td>EFFBE0</td>
<td>EFFBE0</td>
<td>EFFBE0</td>
<td></td>
</tr>
<tr>
<td>Private</td>
<td>ABC_USG_A</td>
<td>USGA_EFF_TL</td>
<td>EFFLE0</td>
<td>EFFLE0</td>
<td>EFFLE0</td>
<td></td>
</tr>
<tr>
<td>Public</td>
<td>ABC_USG_A</td>
<td>USGA_BASE</td>
<td>BasePEO</td>
<td>BasePV0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Public</td>
<td>ABC_USG_A</td>
<td>USGA_EFF_B</td>
<td>EFFBE0</td>
<td>EFFBE0</td>
<td>EFFBE0</td>
<td></td>
</tr>
<tr>
<td>Public</td>
<td>ABC_USG_A</td>
<td>USGA_EFF_TL</td>
<td>EFFVLPE0</td>
<td>EFFVLPE0</td>
<td>EFFVLPE0</td>
<td></td>
</tr>
</tbody>
</table>
This chapter discusses how to use key flexfields in Oracle Fusion applications to access data that is presented by different customers using different combinations of fields, and to customize the presentation of that information to customers in a way that is most appropriate for them.

This chapter includes the following sections:

- Introduction to Key Flexfields
- Completing the Producer Tasks for Key Flexfields
- Completing the Consumer Tasks for Key Flexfields in Reference Mode
- Employing Key Flexfield UI Components on a Page.

25.1 Introduction to Key Flexfields

A key flexfield is a key that is composed of segments, in which one or more segments may have a meaning. The key, or code, uniquely identifies an object such as an account, an asset, a part, or a job, that implementors can configure to validate any way they wish. The definition of a key flexfield provides a list of possible combinations of key flexfield segment values, known as code combinations. Each type of code combination is called a structure. Each structure is identified by a string that is called the structure code. Much like the context values in descriptive flexfields, a key flexfield structure code indicates how database columns are organized to store the code combinations.

25.1.1 Benefits of Key Flexfields

Key flexfields provide a way for Oracle Fusion applications to represent objects such as accounting codes, part numbers, or job descriptions, which combine multiple fields (or segments) into a single object of concatenated segments.

Most customers use codes made up of meaningful segments to identify various business objects. For example, a customer might have a part number "PAD-NR-YEL-8 1/2x14" indicating a notepad, narrow-ruled, yellow, and 14 inches by 8 1/2. Key flexfields enable developers of Oracle Fusion applications to provide customers with flexible code data structures that implementors can set up however they like using key flexfield segments. Key flexfields enable an implementor to customize Oracle Fusion applications to show a customer’s codes any way they want them. For example, a different customer might have a different code for the same notepad, such as "8x14-PD-Y-NR", and the implementor can easily customize Oracle Fusion applications to meet that different need. Key flexfields let developers satisfy different customers without having to reprogram the applications.

You can use key flexfields in many applications. For example, you could use a Part flexfield in an inventory application to uniquely identify parts. Your Part flexfield
could contain such segments as product class, product code, size, color and packaging code. You could define valid values for the color segment, for example, to range from 01 to 10, where 01 means red, 02 means blue, and so on. You could even specify cross-validation rules to describe valid combinations of segment values. For example, products with a specific product code may be available only in certain colors.

25.1.2 How Key Flexfields Are Modeled in Oracle Application Development Framework

Flexfields are modeled as a collection of Oracle Application Development Framework (Oracle ADF) polymorphic view rows, as described in Working with Polymorphic View Rows section in the Developing Fusion Web Applications with Oracle Application Development Framework guide.

In a polymorphic collection, each view row can have its own set of attributes, and all rows have at least one common attribute, the discriminator. The discriminator determines which view row type is used. Given a collection of polymorphic view rows, each row can be a different type. When a polymorphic collection of rows is created, Oracle ADF selects the correctly-typed view definition for the row to be added based on the value of the discriminator attribute.

Key flexfield segments are exposed as view row attributes in the order that they are defined in the flexfield’s metadata. The code combination ID (CCID) and structure instance number (SIN) segments are exposed as attributes in the base view object of the polymorphic collection. Every code combination structure is modeled as an extended view object of the base view object. That is, an extended view object is created for every structure instance number. These extended view objects, which are referred to as subtype view objects, expose the key flexfield segments as subtype-specific view attributes. The structure instance number is exposed as the discriminator attribute of the polymorphic view rows. For more information about code combination IDs and structure instance numbers, see Creating the Combinations Table.

You use the Create Entity Objects wizard to generate the base view object that is based on the key flexfield definition, then create a view link to connect the reference view object (the view object for the database table that the key flexfield extends) and the base view object. You can then use the base view object to add the flexfield to a user interface page. For more information about the generation of base and subtype view objects, see How to Create Key Flexfield Business Components.

Because flexfield view objects are modeled as polymorphic view objects, you can use key flexfield view objects in the same manner that you use any other polymorphic view objects, and they will behave in the same way. This includes support for flexfields in ADF Desktop Integration. For more information, see the Oracle Fusion Middleware Developing Applications with Oracle ADF Desktop Integration guide.

25.1.3 Secondary Usage Feature

A key flexfield configuration can be shared with other product tables through the secondary usage feature. To share a key flexfield configuration with another product table, you include one or all of the primary usage segment columns in that product table. The product table that contains the shared segment columns is referred to as a secondary table. When you work with a primary usage you are working in reference mode. When you work with a secondary usage, you are working in secondary mode.

Note:
Secondary usage is sometimes referred to as a partial usage.
There are two types of secondary usages:

- All-segment secondary usage: In this mode, the secondary table has columns for all of the key flexfield segments.
- Single-segment secondary usage: In this mode, the secondary table has only one key flexfield segment column.

### 25.1.4 Participant Roles

As mentioned in Participant Roles, this guide uses the owner and implementor roles to clarify and group flexfield development activities.

The owner is the developer (or development team) who determines that a particular flexfield is needed or would be useful within a particular Oracle Fusion application, and makes a flexfield available. The owner then incorporates the flexfield into an application. With key flexfields, the owner can be either a producer or a consumer, or can assume both roles.

The producer is the developer who determines that a particular key flexfield is needed or would be useful within a particular application, and makes it available. The producer's product owns the combinations table, which supports that flexfield by storing the key flexfield values.

A consumer incorporates a key flexfield into the application, which is typically different from the producer's application. The consumer typically stores a code combination ID (CCID), which identifies a row in the combinations table, in a product table, and works with the structural and seed data and the business components that have been configured by the key flexfield producer.

An implementor configures a flexfield on behalf of the customer by specifying the structure of the flexfield and specifying the prompt, length, and data type of each flexfield segment.

For more information about owners, producers, consumers, and implementors see Participant Roles. For information about the combinations table, see Creating the Combinations Table.

### 25.1.5 Complete the Key Flexfield Development Process

Before you start to incorporate key flexfields into your application, you must determine whether you should complete the producer portion or the consumer portion of the key flexfield development process.

- If you have determined that a particular key flexfield is needed or would be useful within a particular application, and there is not yet a combinations table to support it, see the producer development tasks covered in Understanding the Key Flexfield Producer Development Tasks.
- If there is already a combinations table for the key flexfield that you want to implement, see the consumer development tasks covered in Understanding the Key Flexfield Consumer Development Tasks.

To incorporate key flexfield secondary usages into your application, see Completing the Development Tasks for Key Flexfields in Secondary Mode.

To employ key flexfield code-combination filters in your application, see Working with Code-Combination Filters for Key Flexfields.
Figure 25-1 provides an overview of producer and consumer roles as they apply to the creation and configuration of the necessary key flexfield business components and associated artifacts. Understanding the Key Flexfield Producer Development Tasks and Understanding the Key Flexfield Consumer Development Tasks define the producer phases and summarize the steps for creating the components and artifacts shown in Figure 25-1.

25.1.5.1 Maintenance Mode and Dynamic Combination Insertion

By default, key flexfield user interface elements do not allow new code combination values entered into the application user interface to be saved. However, you might want to enable the entry of new code combinations in either of the following ways:

- A combination maintenance page enables application implementors and administrators to manage key flexfield code combinations, including the ability to enter new code combinations and update existing code combinations for a flexfield. This is called working in maintenance mode.

- You can enable end users to enter values on an application page that constitute ad-hoc new code combinations, even if the users are not authorized to perform maintenance tasks directly. This is known as dynamic combination insertion.

For example, when entering a transaction, an Oracle General Ledger user can enter a new expense account code combination for an account that does not yet exist. Your application creates the new account by inserting the new combination into the combinations table in the background.
Note:

If you configure the key flexfield for dynamic combination insertion, you must build a maintenance model and a maintenance application module as described in Building a Writable Maintenance Model, and use the setup APIs to register the maintenance application module in the flexfield metadata. For information about the setup APIs, see Registering and Defining Key Flexfields Using the Setup APIs.

The key flexfield producer builds the appropriate models to support maintenance mode and dynamic combination insertion.

25.1.5.2 Cross-Validation Rules and Custom Validation

When you decide to support maintenance mode or dynamic combination insertion for a key flexfield, you can also implement advanced validation capability for the new code combinations that are entered.

Cross-Validation Rules

A cross-validation rule applies a pair of filters to new code combinations that are proposed for a key flexfield by implementors.

At registration time, you must enable the key flexfield for cross-validation. Then you create a maintenance user interface that administrators of your application can subsequently use to define each cross-validation rule as a pair of code-combination filters: one to establish the condition for evaluating the rule, and the other to specify which code combinations are valid under that condition.

Custom Validation Callouts

There are two PL/SQL custom validation callout procedures that can be defined for a given key flexfield: one for application development use, and one reserved for customers. These callouts can be used to enforce any custom validation logic that you want to apply to new code combinations beyond what has been defined for cross-validation rules.

You define custom validation logic with a standard signature for the customer callout. You then register your callout with the key flexfield. The custom validation callout will automatically be called before any new combination is inserted using dynamic insertion in C and PL/SQL.

25.1.5.3 Understand the Key Flexfield Producer Development Tasks

If you have determined that a particular key flexfield is needed within an application, and there are not yet columns in the product table to support it, you must define the necessary metadata and provide the appropriate business components so that flexfield consumers can make use of your flexfield.

To complete the producer development tasks:

1. Develop the key flexfield.
   
   See How to Develop Key Flexfields.
   
   Optionally, you can also do the following at registration time:
   
   • Implement segment labels.
     
     For more information, see How to Implement Key Flexfield Segment Labels.
• Enable cross-validation or register custom validation callout procedures.
  For more information, see How to Implement Cross-Validation Rules and Custom Validation.

2. Create and configure the key flexfield business components.

As shown in Figure 25-1, the producer activities occur in two phases. The first phase produces a writable maintenance model, and the second phase produces a read-only reference model:

• In producer phase 1, you create an updatable entity object for your combinations table and a master view object that is based on the updatable entity object. Next, using the updatable entity object, you create key flexfield business components for a maintenance model and define a view link between the master view object and the key flexfield view objects. Then you create the maintenance application module.

• In producer phase 2, you optionally configure the maintenance application module to accept dynamic combination insertion calls, and implement the appropriate Java class in the user interface to invoke dynamic insertion. You create a read-only reference entity object for your combinations table, and using this entity object, create key flexfield business components for a reference model.

See How to Create Key Flexfield Business Components.

Tip:

After completing this task, you can regenerate the flexfield business components programmatically at runtime to update your key flexfield implementation without manual intervention. For more information, see Regenerating Flexfield Business Components Programmatically.

3. Optionally, share your key flexfield business components with other developers using an ADF library.
  For more information, see How to Share Key Flexfield Business Components.

4. Optionally, build a user interface for key flexfield maintenance.
  For more information, see How to Build a Key Flexfield Maintenance User Interface.

5. Optionally, implement key flexfield advanced features such as code combination constraints or API access to segment labels, or access flexfields from an Excel worksheet using ADF Desktop Integration.
  For more information, see Using Key Flexfield Advanced Features in Reference Mode.

6. Optionally, define, implement, and invoke key flexfield code-combination filters.
  For more information, see Working with Code-Combination Filters for Key Flexfields.

25.1.5.4 Understand the Key Flexfield Consumer Development Tasks

You can incorporate a producer’s key flexfields in your own application. For example, you might have an expenses table that references an account key flexfield in the Oracle General Ledger application.
If your product table already has a foreign-key reference to the key flexfield's combinations table, and the flexfield producer who owns that metadata has provided you with the appropriate business components, you can proceed to incorporate the flexfield into your application. You should have already created an entity object and view object for the product table, which is referred to as the reference table.

To complete the consumer development tasks:

1. As shown in Figure 25-1, create a view link between the view object for the reference table and the polymorphic view objects for the key flexfield’s reference model.
   
   See How to Create Key Flexfield View Links.

2. Nest the key flexfield application module instance in the product application module.
   
   See How to Nest an Instance of the Key Flexfield Application Module in the Product Application Module.

3. Add a key flexfield view object instance to the product application module.
   
   See How to Add an Instance of a Key Flexfield View Object to the Product Application Module.

4. Add your key flexfield to an application page.
   
   See Employing Key Flexfield UI Components on a Page.

5. Configure the key flexfield user interface components.
   
   See How to Configure Key Flexfield UI Components.

After completing these tasks, you can test specified flexfields from JDeveloper using the Run with Flexfields dialog or the Debug with Flexfields dialog. You can also test the flexfields by running the application from an Integrated WebLogic Server instance. For more information, see Testing Flexfields.

After you have completed the key flexfield development process and delivered your application, implementors can use the Manage Key Flexfields task flow to define and configure the structures, structure instances, segments, and segment instances for each key flexfield. This will determine how the flexfield's segments will be populated, organized, and made available to end users within the application.

To make the Manage Key Flexfields task flow available to application implementors and administrators, you register it with Oracle Fusion Functional Setup Manager. For more information, see Integrating Flexfield Task Flows into .

### 25.2 Complete the Producer Tasks for Key Flexfields

To prepare key flexfields for modeling in Oracle JDeveloper, you must ensure that columns for the flexfields you require are defined in your application database. You also might need to define more advanced features such as key flexfield secondary usages, code-combination filters, or the enabling of cross-validation rules and custom validation callout procedures. All of these features require you to modify your application database.
Note:

To incorporate a key flexfield secondary usage into your application, you must have already defined and registered the key flexfield primary usage on which it is based. See Registering and Defining Key Flexfields Using the Setup APIs, then continue to Completing the Development Tasks for Key Flexfields in Secondary Mode.

To employ key flexfield code-combination filters in your application, see Working with Code-Combination Filters for Key Flexfields.

The product table and its key flexfield columns must be registered in the Oracle Fusion Middleware Extensions for Applications (Applications Core) data dictionary before a flexfield can be defined on it. For more information, see Using the Database Schema Deployment Framework.

Any implementation of flexfields in Oracle Fusion applications typically requires application seed data, which is the essential data to enable flexfields to work properly in applications. Flexfield seed data can be uploaded and extracted using the seed data loader.

After you complete the registration process described in How to Develop Key Flexfields, your flexfield seed data consists of the information that you registered for your flexfield, such as the tables and columns reserved for your flexfield.

For information about extracting and loading seed data, see Initializing Oracle Fusion Application Data Using the Seed Data Loader.

25.2.1 How to Develop Key Flexfields

Key flexfields enable you to represent objects such as accounting codes, part numbers, or job descriptions, that combine multiple columns (or segments) into a single object of concatenated segments.

To develop a key flexfield:

1. Create a combinations table that includes the key flexfield segments.
2. Create foreign key columns to associate a product table with the combinations table. The product table is referred to as the reference table.
3. Optionally, include the key flexfield segments in product tables for secondary usages.
4. Optionally, create filter columns for defining which key flexfields the user can filter.
5. Enable the use of flexfield combinations on application pages.
6. Register and define the key flexfield.
7. Optionally enable the multiple structure and data set features.
8. Optionally reuse one or all key flexfield segments in a product table.
9. Register the entity details for each usage.
25.2.1.1 Create the Combinations Table

Each key flexfield must have one corresponding table known as the combinations table.

Note:
The product table and its key flexfield columns must be registered in the Applications Core data dictionary before a flexfield can be defined on it. For more information, see Using the Database Schema Deployment Framework.

The combinations table must have a code combination ID (CCID) column (type NUMBER) that identifies each data row.

The combinations table can have an optional structure instance number (SIN) column (type NUMBER) with generated values that identify different validation sources for a given structure. These generated values are unique within a given flexfield. Multiple SIN values exist for a key flexfield if you elect to define the flexfield with multiple alternate structure instances.

A given structure (arrangement of segments) can have several structure instances. The structure instances share the same arrangement of segments, but use different value sets to validate the segments (for example, one group of value sets for the United States and another for France). Each structure instance is identified by a SIN.

The combinations table might also have a data set number (DSN) column (type NUMBER), but only if you have elected to data set-enable your key flexfield code combinations. A DSN column enables you to tag sets of combination codes with your own numeric IDs. For example, you can use it to stripe (partition) the data into subsets by ORGANIZATION_ID. ADF Business Components supports the DSN by including it as part of the table’s primary key. Your SQL code can then select code combinations from this table using a more qualified primary key.

Note:
Data sets are used by specific application-development teams. If your team does not use data sets, you can ignore the references to DSNs in this guide.

A DSN is not the same thing as a set ID. Set ID partitioning is not supported by flexfields. For information about set IDs, see Managing Reference Data with SetIDs.

The table’s primary key is composed of a combination of the CCID, SIN, and DSN columns depending on the conditions listed in Table 25-1.

Table 25-1  Primary Key Configuration

<table>
<thead>
<tr>
<th>Column</th>
<th>Include in the Primary Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCID</td>
<td>Always</td>
</tr>
<tr>
<td>SIN</td>
<td>When the flexfield is multiple structure-enabled or is multiple structure instance-enabled</td>
</tr>
<tr>
<td>DSN</td>
<td>When the flexfield is DSN-enabled</td>
</tr>
</tbody>
</table>
The combinations table must include the columns listed in Table 25-2. These columns indicate whether a combination is enabled and active. The column names and data types must match exactly.

**Table 25-2  Required Combinations Table Columns**

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENABLED_FLAG</td>
<td>VARCHAR2(1)</td>
<td>A Y value indicates that the combination is enabled. Any other value indicates that the combination is not enabled.</td>
</tr>
<tr>
<td>START_DATE_ACTIVE</td>
<td>DATE</td>
<td>If a date is specified and the current validation date is earlier than the specified date, the combination is not active. There must not be a default database value for this column.</td>
</tr>
<tr>
<td>END_DATE_ACTIVE</td>
<td>DATE</td>
<td>If a date is specified and the current validation date is later than the specified date, the combination has expired. There must not be a default database value for this column.</td>
</tr>
</tbody>
</table>

Include one column for each flexfield segment that you or your customers might wish to customize. You need at least as many columns as the maximum number of segments an end user would ever want in a single key flexfield structure. The columns must be of type VARCHAR2 or NUMBER. If the type is VARCHAR2, the length must be at least 30 characters.

**Tip:**

There are no constraints on how to name the segment columns. However, these columns are typically named using the patterns SEGMENTn_VARCHAR2 and SEGMENTn_NUMBER. This convention makes it easy to identify the key flexfield segments. It also makes it easier to name the columns for secondary usages of the key flexfield.

If the key flexfield defines value attributes, you must include one derived value attribute column of type VARCHAR2 for each value attribute. For more information about value attributes, see How to Implement Key Flexfield Segment Labels.

**Note:**

The combinations table may contain other columns than those described here. If the key flexfield is dynamic insert-enabled, these other columns should either allow null values or they should have default database values.

25.2.1.2 Create Foreign Key Columns to Enable the Use of Flexfield Combinations on Application Pages

To permit the use of flexfield code combinations on different application pages, you must include foreign key references to your combinations table's primary key configuration, as shown in Table 25-1, in other product tables. That way, you can
display or enter valid combinations using forms that are not based on your combinations table. When you build an application that uses key flexfields, you include foreign key references in the product tables wherever you reference the flexfield. The product tables that contain the foreign key references are referred to as reference tables.

**Note:**
A page whose underlying entity objects contain a foreign key reference to the combinations table is referred to as a code-combination reference page, while a page whose underlying entity objects use the combinations table itself is referred to as a combination maintenance page.

### 25.2.1.3 Include Segment Columns in Secondary Tables
You can reuse a key flexfield definition over a product table by including some or all of the key flexfield’s segment columns in the product table. The product table that contains the redefined segment columns is referred to as a secondary table. If a SIN or DSN is used, the secondary table must either include those columns or a column from which the SIN or DSN can be derived.

### 25.2.1.4 Create Filter Columns
You can use the key flexfield code-combination filter feature to represent a subset of combinations. For each filter that you want to include in the application user interface, you define a dedicated column of type `XMLType`. You can define the column in an existing reference table or you can create one or more dedicated tables just to store filter columns.

For more information, see [Working with Code-Combination Filters for Key Flexfields](#).

### 25.2.1.5 Register and Define Key Flexfields Using the Setup APIs
Before you can use a key flexfield in an application, you must first define and register the flexfield using procedures from the `FND_FLEX_KF_SETUP_APIS` PL/SQL package. This package also has procedures for updating, deleting, and querying flexfield definitions.

The definition of a key flexfield includes the following information:

- The code, name, and description of the flexfield.
- The primary usage code (also referred to as the master usage code). This code is typically the same code as the flexfield.
- The name of the combinations database table.
- The names of the database table columns to be used as flexfield segments.
- The name of the CCID column.
- The names of the SIN and DSN columns, if they exist.

**Before you begin:**

Create the combinations table as described in [Creating the Combinations Table](#).

To learn how to generate documentation about using the procedures in the following steps, see [What You May Need to Know About the Key Flexfield Setup API](#).
To register and define a key flexfield:

1. Run the `fnd_flex_kf_setup_apis.create_flexfield(...)` procedure to register the key flexfield and its primary usage.

2. Run the `fnd_flex_kf_setup_apis.create_segment_column_usage(...)` procedure for each segment column to register the segment columns.

3. (Optionally) Register the entity details as described in Registering Entity Details Using the Setup APIs. This step must be completed before you can generate the flexfield usage’s business components.

25.2.1.6 What You May Need to Know About the Key Flexfield Setup API

In the key flexfield development process, you use the `FND_FLEX_KF_SETUP_APIS` PL/SQL package to manage flexfield registration data.

You can learn about the `FND_FLEX_KF_SETUP_APIS` PL/SQL package by running the following command, which produces package documentation and usage examples to the `<db_name>_<user_name>_FND_FLEX_KF_SETUP_APIS_<date>.plsqldoc` file.

```
sqlplus <fusion_user>/<fusion_pwd>@<fusion_db> \
@/ORACLE/fusionapps/atgpf/appcore/db/sql/flex/fnd_flex_pkg_doc.sql \
FND_FLEX_KF_SETUP_APIS
```

25.2.1.7 Enable Multiple Structure, Multiple Structure Instance, and Data Set Features

To enable the multiple structure, multiple structure instance, or data set features for a registered key flexfield, you must run the `enable_feature(...)` procedure from the `FND_FLEX_KF_SETUP_APIS` PL/SQL package. To enable the multiple structure feature or multiple structure instance feature, you provide the SIN column name. To enable the data set feature, you provide the DSN column name.

To learn how to generate documentation about using the `enable_feature(...)` procedure, see What You May Need to Know About the Key Flexfield Setup API.

25.2.1.8 Reuse Key Flexfield Segments in Another Table

Key flexfield secondary usage enables you to capture the values of a key flexfield’s segments in a product table. You can capture all of the flexfield’s segments, or just one.

For information about reusing a key flexfield’s segments, see Completing the Development Tasks for Key Flexfields in Secondary Mode.

25.2.1.9 Register Entity Details Using the Setup APIs

When you build the flexfield business components and create flexfield-specific application module instances, the flexfield modeler requires the following information about the flexfield usage:

- The full class name of the entity object. For the primary usage, this is the entity object that was defined for the combinations table. For a secondary usage, this is the entity object that was defined for the secondary table.

- A prefix from which to derive the names of generated objects.

- The package in which to place the generated business components. Each usage must have its own package name.
You register entity details using the `create_adfbc_usage(...)` procedure from the `FND_FLEX_KF_SETUP_APIS` PL/SQL package.

**Before you begin:**

1. Register the usage as described in *Registering and Defining Key Flexfields Using the Setup APIs*.
2. Ensure that the entity object for the usage’s table exists.

To learn how to generate documentation about using the `create_adfbc_usage(...)` procedure, see *What You May Need to Know About the Key Flexfield Setup API*.

**To register the entity details using the registration application:**

- Run the `fnd_flex_kf_setup_apis.create_adfbc_usage(...)` procedure to register the entity object, package name, and object name prefix for the flexfield usage.

### 25.2.2 How to Implement Key Flexfield Segment Labels

A segment label identifies the purpose of a particular segment in a key flexfield.

Usually an application needs some method of identifying a particular segment for some application purpose such as security or computations. However, because a key flexfield can be customized so that segments appear in any order with any prompts, the application needs a mechanism other than the segment name or order to use for segment identification. Segment labels serve this purpose.

You can think of a segment label as an identification tag for a segment. It identifies a segment that application implementors and administrators should include when customizing the key flexfield. By defining segment labels when you define your key flexfield, you ensure that implementors customize the flexfield to include the segments that your application needs.

For example, the Oracle General Ledger application must be able to identify which segment in the Accounting flexfield contains the primary balance information and which segment contains natural account information. Because you can customize the Accounting flexfield so segments appear in any order with any prompts, Oracle General Ledger needs a segment label to internally specify the correct segment for each purpose. When you define your Accounting flexfield, you must specify which segment labels apply to which segments.

You ensure that the implementor or administrator will define these key segments by defining two segment labels, `GL_BALANCING` and `GL_ACCOUNT`. When customizing your accounting flexfield, the implementor ties the `GL_BALANCING` and `GL_ACCOUNT` segment labels to particular key segments. As the developer, you need not know which key segment becomes the natural account or primary balance segment, because the key flexfield takes care of returning natural account and primary balance information to your application at runtime.

Oracle General Ledger also uses key flexfields that have segment labels identifying the cost center segment (`FA_COST_CTR`), management segment (`GL_MANAGEMENT`), and intercompany segment (`GL_INTERCOMPANY`). Other applications, such as Human Resources, use segment labels as well. Human Resources uses segment labels to control who has access to confidential information in its flexfield segments.
When you use segment labels with a key flexfield, you might also need to define value attributes in which you qualify a value by applying a value attribute to it when the value set is used with a segment that has a segment label.

Note:
For information about retrieving segment label information, see How to Access Segment Labels Using the Java API.

25.2.2.1 Define Key Flexfield Segment Labels

You should define and register segment labels if you want to ensure that the application implementor or administrator customizes your key flexfield to include the segments that your application needs. For example, Oracle General Ledger defines account and balancing segment labels in the Accounting flexfield to ensure that implementors define the account and balancing segments.

When you register a key flexfield, you can define segment labels for it.

Segment labels can be unique, required, or global. You specify a segment label as unique if you want the implementor to tie it to at most one segment of the flexfield. You specify a segment label as required if you want the implementor to tie it to at least one segment. You specify a segment label as global if you want it to apply to all segments. Any key flexfield segment can have any number of segment labels applied.

Table 25-3 shows the results of setting these flags on a segment label in various combinations.

<table>
<thead>
<tr>
<th>Global Flag</th>
<th>Required Flag</th>
<th>Unique Flag</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td>0+ (Zero or more segments)</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>0,1 (Zero or one segment)</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>1+ (One or more segments)</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>1 (Exactly one segment)</td>
</tr>
<tr>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>ALL (All segments; global flag overrides the other flags.)</td>
</tr>
</tbody>
</table>

For example, in the Oracle General Ledger Accounting flexfield, the Account segment label is required and unique because Oracle General Ledger requires one and only one account segment.

You create segment labels using the create_segment_label(...) procedure from the FND_FLEX_KF_SETUP_APIS PL/SQL package.

Before you begin:
Define the key flexfield as described in Registering and Defining Key Flexfields Using the Setup APIs.

To learn how to generate documentation about using the create_segment_label(...) procedure, see What You May Need to Know About the Key Flexfield Setup API.

To define key flexfield segment labels:

- Run the fnd_flex_kf_setup_apis.create_segment_label(...) procedure to register the label and label code for the key flexfield.

25.2.2.2 Use Value Attributes

When you use segment labels with a key flexfield, you might also need to define value attributes.

Every value in a value set has accompanying properties that provide supplemental information about the value, such as a description, an internal code, and start and end dates. In addition to these standard properties, you can further qualify a value by applying a value attribute to it when the value set is used with a segment that has a segment label. There are three types of value attributes:

- **Flexfield value attributes**: The FND_VS_VALUES_B table contains 20 available value attribute columns, called FLEX_VALUE_ATTRIBUTE1 through FLEX_VALUE_ATTRIBUTE20, which are globally defined across all Oracle Fusion applications.

- **Custom value attributes**: The FND_VS_VALUES_B table also contains 10 additional value attribute columns, called CUSTOM_VALUE_ATTRIBUTE1 through CUSTOM_VALUE_ATTRIBUTE10. Customers cannot modify or reassign the standard value attribute columns, but they can use these custom columns for their own implementations of value attributes.

- **SUMMARY_FLAG**: This is a predefined system value attribute, for use with the GL# key flexfield only.

Create value attributes using a procedure from the FND_FLEX_KF_SETUP_APIS PL/SQL package.

Before you begin:

1. Define the key flexfield as described in Registering and Defining Key Flexfields Using the Setup APIs.
2. Define the segment label as described in Defining Key Flexfield Segment Labels.

To define key flexfield segment labels:

- Run the fnd_flex_kf_setup_apis.create_value_attribute(...) procedure to register the value attribute and attribute code for the segment label.

25.2.3 How to Implement Cross-Validation Rules and Custom Validation

Use procedures from the FND_FLEX_KF_SETUP_APIS PL/SQL package to prepare the application database for cross-validation rules and custom validation. When you register a key flexfield in your application database, you can also enable cross-
validation rules and register a customer custom validation callout procedure for the flexfield, so new code combinations entered on a code-combination maintenance page or using dynamic combination insertion can be validated.

At runtime, when a new code combination is entered, the validation APIs are called in the following order:

1. Cross-validation rules
2. Developer custom validation callout procedure
3. Customer custom validation callout procedure

### 25.2.3.1 Implement Cross-Validation Rules

To implement a cross-validation rule for a key flexfield, use a procedure from the FND_FLEX_KF_SETUP_APIS PL/SQL package to enable the flexfield to use cross-validation rules in your application database. Then, build a user interface that administrators can use to maintain their own rule definitions.

**Before you begin:**

Before you can build a user interface for maintaining a key flexfield's cross-validation rules, you must first have created and configured the business components for the key flexfield.

For more information, see How to Create Key Flexfield Business Components.

To learn how to generate documentation about using the FND_FLEX_KF_SETUP_APIS PL/SQL package, see What You May Need to Know About the Key Flexfield Setup API.

**To implement cross-validation rules:**

1. To enable a key flexfield to use cross-validation rules, set the value of the flexfield's CVR_ENABLED_FLAG column in the FND_KF_FLEXFIELDS_B table to Y. This flag is a required VARCHAR2 (1).

   **Note:**
   Setting the value of CVR_ENABLED_FLAG to Y enables support for any cross-validation rules you define for the flexfield. Support for cross-validation is somewhat resource-intensive, so assess each key flexfield to determine if cross-validation is really necessary.

   For example, if the creation of new code combinations for a given key flexfield will be a tightly controlled process that requires organizational oversight, cross-validation might be redundant.

2. To enable administrators to define cross-validation rules that are appropriate for their organizations, develop a runtime maintenance utility that they can use to define and maintain their own rows in a dedicated repository table, FND_KF_CROSS_VAL_RULES, as shown in Table 25-4.
Table 25-4  FND_KF_CROSS_VAL_RULES Cross-Validation Repository Table

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null Allowed?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTERPRISE_ID</td>
<td>NUMBER(18)</td>
<td>No</td>
<td>(Primary key) Enterprise ID.</td>
</tr>
<tr>
<td>STRUCTURE_INSTANCE_ID</td>
<td>NUMBER(18)</td>
<td>No</td>
<td>(Primary key) Structure Instance ID.</td>
</tr>
<tr>
<td>RULE_CODE</td>
<td>VARCHAR2(30)</td>
<td>No</td>
<td>(Primary key) Developer key for this rule.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR2(240)</td>
<td>Yes</td>
<td>Rule description.</td>
</tr>
<tr>
<td>CONDITION_FILTER</td>
<td>XMLTYPE</td>
<td>Yes</td>
<td>Flexfield filter defining where the rule should be applied. A null value means to apply the rule globally.</td>
</tr>
<tr>
<td>VALIDATION_FILTER</td>
<td>XMLTYPE</td>
<td>Yes^1</td>
<td>Flexfield filter defining the validation that must be true.</td>
</tr>
<tr>
<td>ERROR_MSG_APPLICATION_ID</td>
<td>NUMBER(18)</td>
<td>Yes</td>
<td>Message application.</td>
</tr>
<tr>
<td>ERROR_MSG_NAME</td>
<td>VARCHAR2(30)</td>
<td>Yes</td>
<td>Message to display if rule is violated. If a null value, displays the default message.</td>
</tr>
<tr>
<td>ENABLED_FLAG</td>
<td>VARCHAR2(1)</td>
<td>No</td>
<td>Valid values are Y (yes) and N (no).</td>
</tr>
<tr>
<td>START_DATE_ACTIVE</td>
<td>DATE</td>
<td>Yes</td>
<td>Standard start date.</td>
</tr>
<tr>
<td>END_DATE_ACTIVE</td>
<td>DATE</td>
<td>Yes</td>
<td>Standard end date.</td>
</tr>
<tr>
<td>WHO columns^2</td>
<td>Varies depending on the WHO column</td>
<td>Varies depending on the WHO column</td>
<td>Standard WHO columns.</td>
</tr>
</tbody>
</table>

^1 Although the validation filter column must allow null values in the data model, it is still a required value that is logically necessary for cross-validation to work.

^2 For more information about WHO columns, see About Using WHO Column Features.

The primary key for this table is the combination of the ENTERPRISE_ID, STRUCTURE_INSTANCE_ID, and RULE_CODE columns.

The cross-validation rule itself is the combination of a condition filter (when to apply the rule) and a validation filter (how to validate the code combination) in the corresponding XMLType columns of the repository table. These filters are compatible with, and supported by the same infrastructure that supports code-combination filters, as described in Working with Code-Combination Filters for Key Flexfields.
The value of each of these filters should be a logical combination of boolean expressions. At runtime, all filters from the repository table that match the application, key flexfield, and SIN of the newly submitted code combination are retrieved, converted into SQL fragments, and used to validate the proposed code combination.

The condition filter establishes the condition that a proposed new code combination must fulfill to qualify for validation. If it qualifies, the code combination is evaluated against the validation filter. This is demonstrated by Example 25-1.

This means that even though this combination would have failed the validation filter, it is still considered valid because the validation filter was not applied. A code combination fails cross-validation only if it passes the condition filter, but fails the validation filter.

Note:

There are no artificial restrictions on what each filter can contain. If you set the condition filter to a null value, all new code combinations for the flexfield will qualify to be evaluated with the validation filter.

Use the code-combination filter infrastructure to create a separate code-combination maintenance page for each key flexfield that supports cross-validation rules as described in How to Add Code-Combination Filters to Your Application, How to Remove Code-Combination Filters from Your Application, and How to Employ Code-Combination Filters on an Application Page.

Example 25-1 Applying a Cross-Validation Rule

The condition filter specifies a value range for one segment:

\[ \text{segment1} \leq '10' \]

If the condition is met, the validation filter is applied:

\[ (\text{segment2} = '20') \text{ OR } (\text{segment2} = '30') \]

If the proposed code combination is three segments with the following values, the validation will succeed:

\[ \begin{align*}
\text{segment1} &= '8' \\
\text{segment2} &= '30' \\
\text{segment3} &= '70'
\end{align*} \]

If the proposed values are as follows, the condition is not met, and the code combination will not be subject to the validation filter:

\[ \begin{align*}
\text{segment1} &= '12' \\
\text{segment2} &= '40' \\
\text{segment3} &= '70'
\end{align*} \]

25.2.3.2 Implement Custom Validation

To implement custom validation, create and register a PL/SQL validation procedure.
To implement custom validation with a custom validation callout procedure:

1. Write a PL/SQL custom validation procedure.
2. Register the procedure as the customer callout along with the key flexfield to which it will apply.

The PL/SQL validation procedure must have the signature shown in Example 25-2.

**Example 25-2 PL/SQL Validation Procedure Signature**

```plsql
type FLEX_VAL_CTX_RECORD is record (
    VALIDATION_DATE DATE);

procedure MY_VALIDATION_CALLOUT (
    NEW_CODE_COMBINATION in my_comb_table%ROWTYPE,
    VALIDATION_CONTEXT in FLEX_VAL_CTX_RECORD);
```

*my_comb_table* is the name of the combinations table for this key flexfield. When passed, every column in the *NEW_CODE_COMBINATION* record will be populated with the values of the combination that is about to be inserted. Payload columns (columns that are not part of the primary key) in the combinations table that are not related to the flexfield will be passed as null values.

*VALIDATION_CONTEXT* is a record containing any additional usage-specific context that may be useful. Currently this record contains only a *VALIDATION_DATE* field. If there is no validation date, then a null value will be passed for *VALIDATION_DATE*.

The API is expected to raise an exception (with an error message) if validation fails. If an exception is raised, then the dynamic insert operation will be aborted, and the message in the exception displayed to the end user. The log will also record the entire call stack, including the fact that the exception was raised from a custom validation callout procedure. If the API returns without exception, it will be considered a success.

After you have written the custom validation callout procedure, you can register it with the key flexfield. These procedures are registered in the FND_KEY_FLEXFIELDS_B key flexfield registration table as shown in Table 25-5.

**Table 25-5 Key Flexfield Custom Validation Callouts**

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Null Allowed?</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVELOPER_VAL_CALLOUT</td>
<td>VARCHAR2(80)</td>
<td>Yes</td>
<td>PL/SQL validation callout procedure for development use.</td>
</tr>
<tr>
<td>CUSTOMER_VAL_CALLOUT</td>
<td>VARCHAR2(80)</td>
<td>Yes</td>
<td>PL/SQL validation callout procedure for customer use.</td>
</tr>
</tbody>
</table>

To learn how to generate documentation about using the FND_FLEX_KF_SETUP_APIS PL/SQL package, see What You May Need to Know About the Key Flexfield Setup API.

### 25.2.4 How to Create Key Flexfield Business Components

You must define view objects that are based on each key flexfield combinations table. A key flexfield’s base view object has the code combination ID (CCID) column and the optional structure instance number (SIN) column as its only attributes. The SIN, if
applicable, is also the discriminator. For information about base view objects, see How Key Flexfields Are Modeled in .

Figure 25-2 shows a sample configuration of a product view object that uses key flexfields.

The base view object is extended to define view object rows of different structure codes. Each structure code corresponds to a view object definition that includes the appropriate flexfield columns for that structure, in addition to the inherited CCID and SIN columns. Although flexfield view objects carry both SINs and structure codes, only SINs are used to link to the product view object.

**Figure 25-2  Key Flexfields Modeled as ADF Business Components**

If the combinations table has other fixed (nonflexfield) columns, then they are not included in these view objects.

No Java implementation classes are generated for key flexfield view objects. The product view object may or may not have Java implementation classes.

When you create and configure your key flexfield business components, you can decide whether to support maintenance mode (for administrators) or dynamic combination insertion (for end users). For most implementations of a key flexfield, there are two major tasks that you will typically need to complete:

1. **Build a writable maintenance model.**
   
   This model supports building a maintenance mode application, and it supports dynamic combination insertion. It is always required unless you want all end user access to code combinations to be strictly read-only.

2. **Build a read-only reference model.**
   
   This is needed so that you or a consumer of your key flexfield can build a page with a foreign key reference to the combinations table, which is the most likely way that end users will access the key flexfield. This type of page is referred to as a code-combination reference page.

   You can build this model in one of the following ways:

   - **Without dynamic combination insertion support.**
     
     To accomplish this, build your read-only reference model.
• With dynamic combination insertion support.

To accomplish this, you must enable dynamic combination insertion in the maintenance model, and then build the read-only reference model.

Before you begin:

One or more required libraries might have not been automatically included in your project. You must ensure that all required libraries, notably the BC4J Service Runtime, Java EE 1.5 and Java EE 1.5 API libraries, are included.

Using the Create Entity Objects wizard, create application entity objects based on the combinations tables you have defined. Confirm the following:

• At least one customization class is included in the adf-config.xml file. This inclusion serves to ensure correct application behavior. It does not matter which customization class you include.

For information about customization layers, see the "Understanding Customization Layers" section in the Extensibility Guide for Developers.

• These entity objects are directly modeled on the combinations tables; hence they contain the fixed (nonflexfield) columns, if any, along with all of the flexfield columns. In general, all columns should be included.

• The entity objects have primary keys defined.

• The **Persistent** property of every flexfield-related attribute is set to `true`.

• The CCID column is of data type `java.lang.Long`.

• The SIN column, if it exists, is of data type `java.lang.Long`.

• The DSN column, if it exists, is of data type `java.lang.Long`.

• NUMBER type segment columns are of data type `java.math.BigDecimal`.

• VARCHAR2 type segment columns are of data type `java.lang.String`.

• The package name and the object name prefix for each entity object are registered with the ADF Business Components usage to which it will provide data, as described in Registering Entity Details Using the Setup APIs.

### 25.2.4.1 Build a Writable Maintenance Model

A writable maintenance model is the first element required to support a code-combination maintenance page and dynamic combination insertion in your application.

To build a writable maintenance model:

1. Create the maintenance model key flexfield business components.

2. Link the business components to the master view object.

3. Create the maintenance application module.

4. Optionally, implement a method to override the automatic locking of code combinations that are to be inserted or updated, which occurs by default.

#### 25.2.4.1.1 How to Create Key Flexfield Business Components for a Maintenance Model

The first element in a writable maintenance model is a set of business components.
To implement this model, you must select the **Maintenance Mode** checkbox when you encounter it on the Usage Settings page, as described in the following procedure.

**Before you begin:**

1. Create an updatable entity object for the combinations table and add it as an ADF Business Components usage for your key flexfield, as described in *Registering Entity Details Using the Setup APIs*.

   The entity object that you select must allow maintenance operations such as Update or Insert. The entity object class must extend the `oracle.apps.fnd.applcore.oaext.model.KFFMEntityImpl` class, and the entity definition class must extend the `oracle.apps.fnd.applcore.oaext.model.KFFMEntityDefImpl` class.

   **Caution:**
   Disable the delete capability for the combinations table, because the deletion of previously created combinations might invalidate foreign key references. If you want to disallow the use of a combination, then disable the combination instead of deleting it.

2. Create a master view object for the combinations table based on the same updatable entity object.

   This view object typically contains your payload attributes, and should not include flexfield attributes.

   In the master view object, ensure that the CCID attribute’s **Display** control hint is set to **Hide**.

3. Build the project to ensure that the entity objects are available in the project’s classes. The Create Flexfield Business Components wizard relies on what is in the project’s classes.

**To create key flexfield business components:**

1. In the Application Navigator, right-click the project and choose **New**.

2. In the New Gallery, expand **Business Tier**, select **ADF Business Components** and then **Flexfield Business Components**, and click **OK**.

3. On the Flexfield page of the Create Flexfield Business Components wizard, select **Key** from the **Type** dropdown list, as shown in **Figure 25-3**.
4. In the **Application** field, specify the full name of the application to which your key flexfield belongs.
   
   You can browse for the name, and filter by **ID**, **Short Name**, or **Name**.

5. In the **Code** field, specify the code of the key flexfield you want to use.
   
   You can browse for and filter by **Code**.

6. In the **Usage** section, select the table row that contains the primary usage of the key flexfield as it is defined on its combinations table. Every key flexfield has exactly one primary usage.
   
   To identify the primary usage, the **Usage Code** field for this type is typically the same as the flexfield code. The **Table Name** field displays the name of the combinations table, and the text in the **Description** field does not contain the prefix (**Partial**) or (**Partial Single**) in parentheses.
   
   You must select the primary usage in this procedure because you are generating key flexfield business components for the combinations table for your maintenance model.

7. On the **Path** page, select the path of the directory structure in which to create the flexfield business components and click **Next**.

8. On the **Entity Object** page shown in Figure 25-4, select the entity object for the combinations table. It must allow maintenance operations such as update or insert, and include all of the attributes that will be referenced by the flexfield. For the key flexfield primary usage, this includes attributes that represent the CCID, SIN, and segment columns, and the DSN column if it exists in the combinations table.

   **Note:**

   If you select a polymorphic entity object, ensure that the **InheritPersonalization** property for every subtype entity is set to **true**.
9. You might wish to select an entity object for which the key flexfield attributes are defined as transient (not based on database table columns). If you need to do this, then select The names of flexfield attributes match the names of registered columns exactly; do not check the underlying columns.

When a key flexfield entity object attribute is transient, there is no matching underlying column name. When you select this option, the system will match the entity object attribute names to the key flexfield column names, and use the matching attributes to access the flexfield data. Ensure that the entity object has a full set of attributes with matching names before you select this option.

This entity object must be registered under the primary usage. There is no need to register another table for this purpose, even if the entity object is based on some other table. See Registering Entity Details Using the Setup APIs, for more information about registering ADF Business Components usage.

Caution:
The Create Flexfield Business Components wizard is case-sensitive. All column names — and the names of the flexfield entity object attributes associated with them — must be uppercase.

10. Click Next. The Usage Settings page appears.

Because you specified the primary usage of the key flexfield on the Flexfield page of this wizard, this page contains a Maintenance Mode checkbox.

Select Maintenance Mode to build your maintenance model.

11. Click Next. The Naming page displays the entity object’s package name and object name that you registered for the usage, as described in Registering Entity Details Using the Setup APIs. Click Next.

12. On the Summary page, review your choices and click Finish.
The business components generated will replace any existing ones that are based on the same flexfield.

**Note:**

This wizard might fail with a "ClassNotFound" exception message. This indicates that one or more required libraries have not been automatically included in your project, notably the BC4J Service Runtime, Java EE 1.5 and Java EE 1.5 API libraries. You can resolve this issue by manually adding any missing libraries; then you can complete this procedure successfully.

13. Refresh the project to see the newly created flexfield business components in the Application Navigator.

**25.2.4.1.2 How to Link the Master View Object to the Maintenance Model Key Flexfield Business Components**

You must create a flexfield view link from the master view object for the combinations table to the maintenance key flexfield business components. This enables your maintenance user interface to access all of the combinations table columns using the linked view objects for the combinations table entity object.

The master view object and the key flexfield's base view object are linked through the combination of a CCID, and SIN, and if present, a DSN.

**To create a view link for the key flexfield maintenance model:**

1. In the Application Navigator, right-click the project and choose **New**.

2. In the New Gallery, expand **Business Tier**, select **ADF Business Components** and then **Flexfield View Link**, and click **OK**.

3. In the Create Flexfield View Link wizard, on the Name page, provide a package name and a view link name, and click **Next**.

4. In the **Select Source View Object** tree, expand the available objects from the current project and select the master view object for the combinations table, as shown in **Figure 25-5**.
5. In the **Select Destination Flexfield** tree, expand the available flexfield view objects from your project and select your maintenance key flexfield view object as the destination.

6. In the **View Link Accessor Name** field, enter an appropriate name for the view link accessor.

7. Click **Next** to access the Source Attributes page.

**Note:**

For key flexfields in maintenance mode, the Source Attributes page is informational only. The primary key attributes of the source view object will be used to define the view link.

If you see any controls on this page for selecting source attributes, you are not using maintenance mode business components. Return to **Building a Writable Maintenance Model** and re-create your maintenance mode business components according to the instructions.

8. Click **Finish** to go to the Summary page.

**Note:**

You can skip the Properties page because view link-specific properties are not supported.

9. On the Summary page, review the summary, then click **Finish**.

### 25.2.4.1.3 How to Create the Maintenance Application Module

You must create the maintenance application module for the key flexfield. The application module contains the combination view object, the maintenance model view link, and the key flexfield's application module that was created when you created the business components in **How to Create Key Flexfield Business Components for a Maintenance Model**.

---

**Figure 25-5  Create Flexfield View Link Wizard — View Objects Page**

![Create Flexfield View Link Wizard](image-url)
For more information about creating application modules and nesting application model instances, see the Implementing Business Services with Application Modules chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

Before you begin:

1. Create the business components as described in How to Create Key Flexfield Business Components for a Maintenance Model.

2. Create the view link as described in How to Link the Master View Object to the Maintenance Model Key Flexfield Business Components.

To create the maintenance application module:

1. In the Application Navigator, right-click the project and choose New.

2. In the New Gallery, expand Business Tier, select ADF Business Components and then Application Module, and click OK.

3. In the Create Application Module wizard, on the Name page, provide a package name and an application module name, and click Next.

4. In the Data Model page, move the master view object for the combinations table and the maintenance model view link to the Data Model list, as shown in Figure 25-6.

**Figure 25-6  Create Application Module Wizard — Data Model Page**

Tip:
The object name prefix and package name are used to name the flexfield business components, and are defined in the database along with the key flexfield.

5. Click Next. The Application Modules page appears.
6. On the Application Modules page, in the Available tree, move the appropriate key flexfield application modules to the Selected list, as shown in Figure 25-7. Include the key flexfield application module that was created when you created the business components in How to Create Key Flexfield Business Components for a Maintenance Model.

**Figure 25-7  Create Application Module Wizard — Application Modules Page**

![Image of Application Modules page from Create Application Module Wizard]

**Note:**
For each key flexfield, only one instance of the application module is needed. For example, even though two view links may have been created to access the same flexfield, only one instance of the flexfield application module is needed in the product application module.

7. When you complete the Create Application Module wizard, right-click the new application module instance and choose Run to test it.

**Note:**
The maintenance application module must have a configuration named appmodule_nameLocal. By default, this is created for you. For example, if the application module is called MyKffMaintAM, then a configuration named MyKffMaintAMLocal must exist.

**Tip:**
You can publish a key flexfield application module instance as a web service. For more information about creating and testing a key flexfield service interface, see How to Publish Key Flexfield Application Modules as Web Services.
25.2.4.1.4 How to Manage Code Combination Locking

The `oracle.apps.fnd.applcore.oaext.model.KFFMEntityImpl` class, which is extended by your code combination entity object class, enables automatic code combination locking by default. The method `doDML(int, TransactionEvent)` is overridden to lock the code combination to be inserted or updated. The lock is removed when the transaction is committed or rolled back.

If you wish to completely overwrite `doDML` with your own implementation, you can turn the automatic locking off by calling `setAutoCombinationLockEnabled(false)`, then calling `lockCombination(DBTransaction)` on your own. For more information, see the Java documentation for `oracle.apps.fnd.applcore.oaext.model.KFFMEntityImpl`.

25.2.4.2 Enable Dynamic Combination Insertion

This task is necessary only if you want to permit end users to create new code combinations spontaneously on an application page. You must have already built a writable key flexfield maintenance model. For more information, see How to Create Key Flexfield Business Components.

To enable this feature, you define an application module that you configure for dynamic combination insertion, then implement the appropriate Java class in the user interface. You can create a basic implementation of dynamic combination insertion under the simplest conditions, create a more sophisticated version that includes added combination attributes, or, if custom validation procedures or cross-validation rules are registered with the flexfield, create a version that makes information available to the custom validation procedures.

25.2.4.2.1 Enable Dynamic Combination Insertion

To enable dynamic combination insertion, the key flexfield must be set to allow dynamic combination insertion, and the full name of the maintenance application module that will implement `KFFCombinationCreator` must be registered with the key flexfield.

To enable dynamic combination insertion:

1. Issue the following SQL update statement to enable dynamic combination insertion:

   ```
   update fnd_kf_flexfields_b
   set dynamic_combo_creation_flag = 'Y'
   where application_id = :app_id
   and key_flexfield_code = :kff_code
   ```

   Set the `:app_id` to the `application_id` that was specified when the flexfield was created, and set `:kff_code` to the flexfield's `key_flexfield_code`. For more information, see What You May Need to Know About the Key Flexfield Setup API.

2. Issue the following SQL statement to set the name of the application module to be used for dynamic combination insertion:

   ```
   update fnd_kf_flexfields_b
   set application_module_name = 'fully qualified name of application module'
   where application_id = :app_id
   and key_flexfield_code = :kff_code
   ```
The name of the application module must be fully qualified; for example, mycompany.myproduct.flex.kff1.applicationModule.Kff1AM.

Set the :app_id to the application_id that was specified when the flexfield was created, and set :kff_code to the flexfield's key_flexfield_code. For more information, see What You May Need to Know About the Key Flexfield Setup API.

### 25.2.4.2.2 Insert a Code Combination — the Simplest Case

In the simplest case, you need only replace the existing base object class, oracle.apps.fnd.applcore.aoext.model.OAApplicationModuleImpl, with oracle.apps.fnd.applcore.aoext.model.KFFCombinationCreatorImpl. KFFCombinationCreatorImpl extends OAApplicationModuleImpl.

This implementation is possible only under the following conditions:

- You have no custom Java application module class for this application module.
- Only one key flexfield application module instance is nested in this application module, and that nested instance is the one that represents the key flexfield of interest.
- You do not need to update any columns of the combinations table, including the value attribute columns.

### 25.2.4.2.3 Insert a Code Combination with Added Combination Attributes

To insert a code combination with added combination attributes, you implement the KFFCombinationCreator Java class in the maintenance application module, and you create a Java implementation of the maintenance application module. You can optionally initialize the columns.

1. In the maintenance application module, implement the Java class oracle.apps.fnd.applcore.aoext.model.KFFCombinationCreator from the oracle.apps.fnd.applcore.aoext.model package.

   The class has only one method defined:

   ```java
generic void createKeyFlexfieldCombination(Long sin, Long dsn, List<Object> segValues);
```

   This method has the following parameters:

   - **sin**: The structure instance number. This should be null if this key flexfield does not allow multiple structures.
   - **dsn**: The data set number. This should be null if this key flexfield does not use data set numbers.
   - **segValues**: A read-only list of segment values.

   If an error occurs during creation, throw the exception FlexfieldJboException.

2. Generate a Java application module class to extend your existing base object class. By default, the base class is OAApplicationModuleImpl from the oracle.apps.fnd.applcore.aoext.model package. An example of an application module class is shown in Example 25-3.
By default, JDeveloper creates accessor methods (such as getMyKffAM1()) to all of your nested application modules. You can use these accessor methods to access the key flexfield application module, as shown in bold in Example 25-3.

If you do not need to update any combination attributes, then the implementation in the example is sufficient; otherwise, you can use the KFFCombinationAttributes object to update the value attribute columns, or use the master view object to update any other columns of the combinations table, as described in the next steps.

3. Optionally, initialize value attribute columns using KFFCombinationAttributes.

The KFFCombinationAttributes object enables you to:

- Get the segment values and their value-attribute values.
- Get the default values of the value attributes used.
- Get the list of value attribute codes for a label used in this flexfield.
- Get the current value of a value attribute column of the combinations table.
- Update a value attribute column of the combinations table.

By default, the standard value attribute columns such as START_DATE_ACTIVE, END_DATE_ACTIVE, and ENABLED_FLAG are initialized in the insertCombination call. The ENABLED_FLAG is initialized to Y. The START_DATE_ACTIVE value is set to the maximum of the START_DATE_ACTIVE values for the segments, or NULL if all values are null. The END_DATE_ACTIVE value is set to the minimum of the END_DATE_ACTIVE values for the segments, or NULL if all values are null. You have full access to these value-attribute values and can update these columns of the combinations table if you wish.

Example 25-4 demonstrates how to use the KFFCombinationAttributes object to access these values and update the value attribute columns of the combinations table.

4. If you want to initialize other columns of the combinations table, first include them in the master view object. After insertCombination is called, the new entity will be available to the master view object as well, as shown in Example 25-5, which shows an alternative version of MyKffMaintenanceAM.java.

**Example 25-3 Implementing KFFCombinationCreator**

```java
public class MyKffMaintenanceAM
    extends OAApplicationModuleImpl
    implements KFFCombinationCreator
{
    /**
     * Container's getter for MyKffAM1.
     *
     * @return MyKffAM1
     */
    public ApplicationModuleImpl getMyKffAM1()
    {
        return (ApplicationModuleImpl) findApplicationModule("MyKffAM1");
    }

    public void createKeyFlexfieldCombination(
```
Example 25-4 Using the KFFCombinationAttributes Object

```java
public void createKeyFlexfieldCombination(
    Long sin, Long dsn, List<Object> segValues)
{
    KFFCombinationAttributes combAttrs =
        ((KFFMApplicationModuleImpl)
          getMyKffAM1()).insertCombination(sin, dsn, segValues);
}

final String myLabel = "MY_LABEL";
// Get the segment values and their value-attribute values for the label.
List<FlexfieldSegmentValue> segValueList = combAttrs.getSegmentValues(myLabel);
// Loop through each segment value.
for (FlexfieldSegmentValue segValue: segValueList)
{
    // Get the segment code if needed.
    System.out.println("SegmentCode = " + segValue.getSegmentCode());
    // Get the segment value if needed.
    System.out.println("SegmentValue = " + segValue.getValue());
    // Iterate through each value attribute code for the label if needed.
    Iterator<String> it = combAttrs.getValueAttrCodeIterator(myLabel);
    while (it.hasNext())
    {
        String valAttrCode = it.next();
        FlexfieldSegmentValue.ValueAttributeValue valAttrValue =
            segValue.getValueAttributeValue(valAttrCode);
        // Value attribute code is also available in the value object.
        System.out.println("  ValueAttrCode = "
                           + valAttrValue.getValueAttributeCode());
        // Get the default value of the value attribute.
        System.out.println("  ValueAttrDefaultValue = "
                           + valAttrValue.getDefaultValue());
        // Get the value of the value attribute.
        System.out.println("  ValueAttrValue = "
                           + valAttrValue.getValue());
    }
}

System.out.println();

final String myValueAttrCode = "MY_VALUEATTRIBUTE1";
// Get the current combination value attribute.
System.out.println("\t" + myLabel + ":" + myValueAttrCode + " = "
                           + combAttrs.getValueAttribute(myLabel, myValueAttrCode));

// Update the combination value attribute.
Map<String, Object> valueMap = new HashMap<String, Object>(1);
valueMap.put(myValueAttrCode, "N");
combAttrs.setValueAttributes(myLabel, valueMap);

System.out.println();
```
/** Dealing with standard value attributes. ***/

// Get the segment values with the standard value attributes.
List<FlexfieldSegmentValue> segValueListStd = combAttrs.getSegmentValues();
// Loop through each segment value.
for (FlexfieldSegmentValue segValue: segValueListStd)
{
    System.out.println("SegmentCode = " + segValue.getSegmentCode());

    // Get the START_DATE_ACTIVE attribute.
    FlexfieldSegmentValue.ValueAttributeValue startDateActive =
    segValue.getValueAttributeValue(
        FlexfieldSegmentValue.VALUE_ATTR_START_DATE_ACTIVE);
    // Get the END_DATE_ACTIVE attribute.
    FlexfieldSegmentValue.ValueAttributeValue endDateActive =
    segValue.getValueAttributeValue(
        FlexfieldSegmentValue.VALUE_ATTR_END_DATE_ACTIVE);
    // Get the ENABLED_FLAG attribute.
    FlexfieldSegmentValue.ValueAttributeValue enabledFlag =
    segValue.getValueAttributeValue(
        FlexfieldSegmentValue.VALUE_ATTR_ENABLED_FLAG);

    System.out.println("  StartDateActive = " + startDateActive.getValue());
    System.out.println("  EndDateActive = " + endDateActive.getValue());
    System.out.println("  EnabledFlag = " + enabledFlag.getValue());
}
System.out.println();

// Get the current combination start date.
System.out.println("StartDateActive = " + combAttrs.getValueAttribute(null,
    FlexfieldSegmentValue.VALUE_ATTR_START_DATE_ACTIVE));
// Get the current combination end date.
System.out.println("EndDateActive = " + combAttrs.getValueAttribute(null,
    FlexfieldSegmentValue.VALUE_ATTR_END_DATE_ACTIVE));
// Get the current combination enabled flag.
System.out.println("EnabledFlag = " + combAttrs.getValueAttribute(null,
    FlexfieldSegmentValue.VALUE_ATTR_ENABLED_FLAG));

// You can update the standard value attributes by calling
// combAttrs.setValueAttributes(Map).

Example 25-5 Calling insertCombination to Make the New Entity Available to the
Master View Object

public void createKeyFlexfieldCombination(
        Long sin, Long dsn, List<Object> segValues)
    {
            KFFCombinationAttributes combAttrs =
            ((KFFMApplicationModuleImpl)
                getMyKffAM1()).insertCombination(sin, dsn, segValues);

            // In this example, the key flexfield allows multiple structures.
            // The order of the key values must match the order of the keys;
            // see ADF Business Components Java documentation for more details.
            Key keyValues = new Key(new Object[] {
                combAttrs.getCodeCombinationID(),
                sin});
    }
// getMyKffComboAttrVO() is generated when a view object instance named
// "MyKffComboAttrVO" is present. You can always call
// findViewObject to find the
// view object if no accessor method is available.
OAViewObjectImpl vo = getMyKffComboAttrVO();
Row[] rows = vo.findByKey(keyValues, 1);
if (rows != null && rows.length == 1)
{
    rows[0].setAttribute("MyCombinationAttr1", "Y");
} else
{
    throw new FlexfieldJboException("Unable to find the newly created combination: CCID = "+
combAttrs.getCodeCombinationID()
+ ", SIN = "+sin);
}

25.2.4.2.4 Insert a Code Combination that Uses Custom Validation Procedures or Cross-Validation Rules

If custom validation procedures or cross-validation rules are registered with the flexfield, you must create a Java class for the maintenance application module that implements the KFFCombinationCreatorProxy class. This makes the information in the KFFCombinationCreatorProxy.Context object, such as the validation date, available to the custom validation procedures.

1. In the maintenance application module, generate a Java class that implements KFFCombinationCreatorProxy from the oracle.apps.fnd.appcore.oaext.model package and extends your existing base object class. By default, the base class is OAApplicationModuleImpl from the same package. An example of an application module class is shown in Example 25-3.

Note:

KFFCombinationCreatorProxy is a sub-interface of KFFCombinationCreator.

2. Optionally, initialize value attributes, as described in Step 3 of Inserting a Code Combination with Added Combination Attributes.

3. Optionally, initialize other columns of the combinations table, as described in Step 4 of Inserting a Code Combination with Added Combination Attributes.

Example 25-6 Implementing KFFCombinationCreatorProxy

public class MyKffMaintenanceAM extends OAApplicationModuleImpl implements
KFFCombinationCreatorProxy
{
    /**
     * Container's getter for MyKffAM1.
     *
     * @return MyKffAM1
     */
public ApplicationModuleImpl getMyKffAM1()
{
    return (ApplicationModuleImpl) findApplicationModule("MyKffAM1");
}

@Override
public void createKeyFlexfieldCombination(
    Long sin, Long dsn, List<Object> segValues)
{
    // Delegated to the method that takes "context".
    this.createKeyFlexfieldCombination(sin, dsn, segValues, null);
}

@Override
public void createKeyFlexfieldCombination(final Long sin, final Long dsn,
    final List<Object> segValues,
    final Context context)
{
    KFFCombinationAttributes combAttrs =
    {
        ({KFFMApplicationModuleImpl) getMyKffAM1()}.insertCombination{
            sin, dsn, segValues, context);
    }
}

### 25.2.4.3 Build a Read-Only Reference Model

A set of business components that constitute a read-only reference model is needed so that you or a consumer developer can build a code-combination reference page, which is typically a transaction page for a product table that has a foreign key reference to the combinations table.

---

**Note:**

If you want to permit end users to create new code combinations spontaneously on an application page, you must have already built a writable key flexfield maintenance model and have enabled dynamic combination insertion.

For more information, see **How to Create Key Flexfield Business Components**.

---

**Before you begin:**

Create a read-only entity object for your combinations table and add it as an ADF Business Components usage for your key flexfield, as described in **Registering Entity Details Using the Setup APIs**.

**To create key flexfield business components for a read-only reference model:**

1. Complete Step 1 through Step 8 in **How to Create Key Flexfield Business Components for a Maintenance Model**.

2. On the Entity Object page, expand the tree of available models and select the read-only entity object that you created for the combinations table.

    The entity object you select must include all of the attributes that will be referenced by the flexfield. For the key flexfield primary usage, this includes
attributes that represent the CCID, SIN, and segment columns, and the DSN column if it exists in the combinations table.

3. Complete Step 9 through Step 13 in How to Create Key Flexfield Business Components for a Maintenance Model.

25.2.5 How to Share Key Flexfield Business Components

Sharing flexfield business components is just like sharing any other ADF Business Components objects. You can share the objects through an ADF Library JAR file. The developers then can import the business components that are contained in the JAR file.

For more information, see the Packaging a Reusable ADF Component into an ADF Library section in the Developing Fusion Web Applications with Oracle Application Development Framework.

25.2.5.1 Create an ADF Library JAR File

If you are the owner of the flexfield business components that you want to share, then you can create a JAR file containing those business components. Generally, an entire JDeveloper project is deployed as an ADF Library JAR file.

Create your shared ADF library containing the business components from the read-only reference model you just built, then add the business components from the writable maintenance model as well.

To create an ADF Library JAR file:

1. Right-click the project you wish to share and select Project Properties from the menu.

2. Select Deployment.

3. If a deployment profile is already listed, you can verify that it is for an ADF Library JAR file. Open the profile for editing and observe the window title to confirm that it says "Edit JAR Deployment Profile Properties.”

If you do not already have an appropriate deployment profile, you can create one:

a. In the Application Navigator, right-click the project and choose New.

b. In the New Gallery, expand General, select Deployment Profiles and then ADF Library JAR File, and click OK.

c. In the Create Deployment Profile — ADF Library JAR File dialog, enter a name for the profile and click OK.

4. Right-click the project you wish to share and select Deploy > deployment_profile_name > To JAR File.

The JAR file is created in your project's deploy directory as deployment_profile_name.jar. You can send it to other developers for use in their projects.

25.2.5.2 Import Business Components from an ADF Library

After an ADF Library JAR file has been created by one developer, another developer can import the business components that are contained in the file.
For more information, see the Adding ADF Library Components into Projects section in the Developing Fusion Web Applications with Oracle Application Development Framework.

To Import business components from an ADF library:

1. Obtain an ADF Library JAR file from another developer and save it to an accessible directory for importing.
2. Right-click the project where you want to import the components and select Project Properties.
4. Click Import, then go to the location of the ADF Library JAR file.
5. Select the ADF Library JAR file and click Open.

The business components in the JAR file are imported into your project.

25.2.6 How to Build a Key Flexfield Maintenance User Interface

Use the business components from a writable maintenance model to build a key flexfield code-combination maintenance page. Building a maintenance page is fairly straightforward. If you have already inserted a key flexfield component into a page in either form or table layout, building a maintenance page follows a similar pattern.

To enable end users to select key combinations for new rows, ensure that you provide a default value for every Structure Instance Number attribute.

25.2.6.1 Build a Key Flexfield Code-Combination Maintenance Page

Build a key flexfield code-combination maintenance page using objects from the maintenance application module that you created over the combinations table.

Before you begin:

Create a maintenance application module for the combinations table and a writable maintenance model for the flexfield usage as described in Building a Writable Maintenance Model.

To build a code-combination maintenance page:

1. From the Data Controls panel, expand the maintenance application module that you created for the combinations table.
2. Drag the master view object from the application module onto the page, and add it as either a form or a table.
3. Select the key flexfield view object and drag it onto the page. Drop the flexfield view object onto the form or table that you just created.

For more information, see Employing Key Flexfield UI Components on a Page.

25.2.6.2 Ensure Proper Handling of New Rows

When a new row that contains key flexfield columns is added on the code-combination maintenance page, every Structure Instance Number attribute must contain a valid value, so that the key flexfield user interface can be rendered with appropriate structures. Without default structures, end users will not be able to select segment values for the new row.
You can prevent application errors by defining a default value for each Structure Instance Number. Edit the Structure Instance Number attribute in either the entity object or the view object for the reference table, and do one of the following:

- If the Structure Instance Number is static, set the Value Type to Literal, and specify the static value as the default.
- If the Structure Instance Number is dynamic, set the Value Type to Expression, and enter a Groovy expression to retrieve the appropriate Structure Instance Number value and set it as the default.

---

**Note:**

Use the defaultSIN attribute in the JSPX file to define the default Structure Instance Number value for situations where no rows exist.

---

**Caution:**

The Structure Instance Number value of the first row of a user interface table determines the column structure to be used for the table. For any additional row that contains a different Structure Instance Number, the columns that are not also part of the first row's structure will not render in the table. If there are no rows, the value of the defaultSIN tag attribute from the JSPX file, if set, determines the column structure.

For an Applications Table component, if the end user deletes all rows from the table, your application can again set a new default Structure Instance Number value for the first new row, and the column structure corresponding to that Structure Instance Number will be the valid structure for the table.

For an ADF Table, the column structure (determined by the initial Structure Instance Number value) cannot be changed after the table has been created, even if all rows are deleted.

---

For more information about setting attribute defaults, see the discussion about defining default values in the Setting Attribute Properties section of the Developing Fusion Web Applications with Oracle Application Development Framework.

### 25.2.7 What Happens at Runtime: Creating New Combinations

At runtime, an instance of the registered application module is created. Whenever a new combination must be created, the following happens:

1. The createKeyFlexfieldCombination method is invoked. When KFFApplicationModuleImpl.insertCombination is called in the implementation, a lock is created to ensure that no one else can insert the same combination.

2. The transaction of the registered application module is committed or rolled back. The lock is always removed.

3. One of the following occurs:
   - If no exception is thrown, the transaction is committed.
• If any exception occurs, the transaction is rolled back.

25.3 Complete the Consumer Tasks for Key Flexfields in Reference Mode

You can reference flexfields from another (producer) application into your (consumer) application. The consumer tasks for a key flexfield primary usage (also referred to as reference mode) are:

1. Create a view link from your (consumer) product view object to the producer's key flexfield view object.
2. Nest the producer's key flexfield application module instance in the (consumer) application module for the application.
3. Add a key flexfield view object instance to the (consumer) application module for your application.

After you complete the consumer tasks, you can incorporate the key flexfield UI components into your application as described in Employing Key Flexfield UI Components on a Page.

25.3.1 How to Create Key Flexfield View Links

If a product view object references a producer’s key flexfield, you must link the product view object to the key flexfield’s base view object to create a master-detail hierarchy. You use the Create Flexfield View Link wizard two link the two view objects. The product view object and the key flexfield’s base view object are linked through the combination of a CCID, a SIN, and if present, a DSN. The key flexfield view object can have many incoming view links from various product view objects, because a key flexfield is usually referenced by many product tables. For example, an ExpenseLines view object might have a foreign key reference to the GLKff view object. For more information about view links and master-detail hierarchies, see the Defining SQL Queries Using View Objects chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

By default, when a value set is security-enabled, any key flexfield code combination segment that uses that value set will automatically be secured. Security rules defined on the value set are propagated automatically to the combinations table, and also to any product table that references the combinations table. This means that when an end user does a search on the product table, the results shown are limited to the data referencing the code combination entries to which the user has access. You can add a custom property to the view link to disable the propagation of the security rules to the product table.

Before you begin:

• Ensure that the flexfield library JAR files that were created by the producer team have been added to your project. For more information about library JAR files, see Creating an ADF Library JAR File and Importing Business Components from an ADF Library.

• You should have already created an entity object and a view object for the reference table (the product table with a foreign key reference to the combinations table). You use the reference table’s view object for the source of the view link.

Ensure that the view object does not include flexfield attributes such as SEGMENT1 VARCHAR2, SEGMENT2 NUMBER, and so on. Ensure that you include the attributes that are needed for the foreign key reference, such as CCID,
To create a key flexfield view link:

1. In the Application Navigator, right-click the project and choose New.
2. In the New Gallery, expand Business Tier, select ADF Business Components and then Flexfield View Link, and click OK.
3. In the Create Flexfield View Link wizard, on the Name page, provide a package name and a view link name, and click Next.
4. In the View Objects page, in the Select Source View Object tree, expand the available objects from your current project and select a source view object.
5. In the Select Destination Flexfield tree, expand the available flexfield view objects from your project and select the key flexfield’s base view object with which you want to associate the source view object.
6. In the View Link Accessor Name field, enter an appropriate name for the view link accessor.
7. Click Next. The Source Attributes page appears, as shown in Figure 25-8.

Figure 25-8  Create Flexfield View Link Wizard — Source Attributes Page for Key Flexfields

8. From the Code-Combination ID dropdown list, select the source attribute that corresponds to the CCID of the destination key flexfield.
   The CCID must be mapped to type java.lang.Long.
9. If your destination key flexfield supports multiple structure instances, the Structure Instance Number dropdown list appears on the Source Attributes page. You must specify a structure instance as an additional source attribute. From the dropdown list, select the source attribute that corresponds to the SIN of the destination key flexfield.
The SIN must be mapped to type `java.lang.Long`.

---

**Note:**

The source attribute must be an entity attribute that is either persistent or is SQL-derived.

---

10. If your destination key flexfield supports multiple structure instances and is data set-enabled, the **Data Set Number** dropdown list appears on the Source Attributes page. You must specify a data set as an additional source attribute. From the dropdown list, select the source attribute that corresponds to the DSN of the destination key flexfield.

    The DSN must be mapped to type `java.lang.Long`.

11. Click **Finish** to go to the Summary page.

    **Note:**

    You can skip the Properties page because view link-specific properties are not supported.

12. On the Summary page, review the summary, then click **Finish** to create the view link.

13. Optionally, disable the automatic propagation of value set security rules to the product table by adding a custom property to the view link between the product view object and the key flexfield view object, as follows:

    ```xml
    <propertyname="FND_ACFF_MasterSecuredByFlexfield" Value="false"/>
    ```

---

### 25.3.2 How to Nest an Instance of the Key Flexfield Application Module in the Product Application Module

Use the overview editor for the product application module to nest the application module instance for the key flexfield. This is the application module instance that was created when you created the flexfield business component and was named using the prefix that you specified when you defined the usage’s entity details. The nested key flexfield application module instance shares the same transaction and entity object caches as the application module.

**Before you begin:**

You should have already created a product application module.

**To nest an instance of the key flexfield application module in the product application module:**

1. In the Application Navigator, double-click the product application module.

2. In overview editor, click the **Data Model** navigation tab.

3. On the Data Model Components page, expand the **Application Module Instances** section and, in the **Available** list, select the key flexfield application module.

    The **New App Module Instance** field below the list shows the name that will be used to identify the next instance that you add. You can change this name.
4. With the desired application module selected, move the key flexfield application module to the **Selected** list.

### 25.3.3 How to Add an Instance of a Key Flexfield View Object to the Product Application Module

You must add a flexfield view object instance that reflects the master-detail hierarchy of the view link that you created in *How to Create Key Flexfield View Links* to the product application module. You can use the data model that the application module overview editor displays to create the master-detail hierarchy of view instances. The master view object is the view object for the reference table (the product table with a foreign key reference to the combinations table), and the detail view object is the view object for the flexfield. For example, if you created a view link from the `ExpenseLines` view object to the `GLKff` view object, then `ExpenseLines` is the master and `GLKff` is the detail.

For more information about creating a hierarchy of view instances, see the "Adding Master-Detail View Object Instances to an Application Module" section in the *Developing Fusion Web Applications with Oracle Application Development Framework*.

**Before you begin:**

1. Add an instance of the view object for the reference table to the product application module, as described in the "How to Add a View Object Instance to an Application Module" section in the *Developing Fusion Web Applications with Oracle Application Development Framework*.

2. Create a view link between the reference table view object and the key flexfield view object as described in *How to Create Key Flexfield View Links*.

**To add an instance of a key flexfield view object to the product application module:**

1. In the Application Navigator, double-click the product application module.

2. In the overview editor, click the **Data Model** navigation tab.

3. Expand the **View Object Instances** section.

4. If an instance of the reference table view object does not appear in the **Data Model** list, then select it in the **Available View Objects** list and move it over. This is the master view object instance.

5. In the **Data Model** list, select the instance of the view object for the reference table (the master view object instance) so that it appears highlighted. This will be the target of the detail flexfield view instance that you will add.

6. In the **Available View Objects** list, expand the view object for the reference table and move the nested key flexfield view object (the detail view object) to the **Data Model** list. The flexfield view instance appears nested under master view instance, as shown in *Figure 25-9*. 
25.4 Employ Key Flexfield UI Components on a Page

After a key flexfield's UI components have been made available to your application, you can incorporate the flexfield into UI pages and query search forms. You can then configure the various aspects of an incorporated key flexfield UI component to customize the behavior of the flexfield as a whole, or on a segment-by-segment basis.

25.4.1 How to Employ a Key Flexfield Component on a Page

To employ a key flexfield UI component on an application page, add the flexfield to a form component or a table component, then configure the properties of the flexfield.

Note:

This section assumes you are using the data-first method of adding flexfields to your application, in which you build the data model first, then create the user interface by dragging data controls onto a page. The UI-first method is also available, but is not documented here.

Key flexfields can be implemented on the following types of pages:

- **A page with a foreign key reference to the combinations table**

  The database table (or view) for this type of page contains a foreign key reference to a combinations table that contains the actual flexfield segment columns. You create a page with a foreign key reference if you want to use your page to manipulate rows that contain code combination IDs. The database table is referred to as the reference table, and the page is referred to as the code-combination reference page.

  The primary purpose of code-combination reference pages is generally unrelated to the fact that some fields might be key flexfields. That is, the purpose of the page is to accomplish whatever business function is required (such as entering orders, receiving parts, and so on). You might have many code-combination reference pages that use a given key flexfield.
• A page with secondary usage of a key flexfield
  You can invoke the secondary usage feature of key flexfields on a page. Secondary usage occurs when one or all segments of a key flexfield that have already been defined for a combinations table are also defined for a product table. In this way, you can reuse a key flexfield definition for a product table as if it were a descriptive flexfield.

• A code-combination maintenance page
  The only purpose of a code-combination maintenance page is to create and maintain code combinations. This page is typically built by the producer. The combinations table (or a view of it) is the base table of this page and contains all the key flexfield segment columns. The combinations table also contains a unique ID column. For information about creating a code-combination maintenance page, see How to Build a Key Flexfield Maintenance User Interface.

  A typical application has one and only one code-combination maintenance page. An application might not have a code-combination maintenance page if it does not support maintenance mode for administrators.

• A page containing a search form
  An advanced search form enables end users to define criteria to search for metadata in the application’s master view object and its linked key flexfield view object. Users can select which attributes of the key flexfield view object to use as criteria. See How to Incorporate Key Flexfields into a Query Search Form for information about using key flexfields in a search form.

  Note:
  You cannot use a key flexfield in a tree table component.

In a typical application, you would have one code-combination maintenance page that maintains the key flexfield, where the key flexfield is the representation of an entity in your application. You would also have one or more code-combination reference pages with foreign key references to the same combinations table. For example, in an order entry/inventory application, you might have a code-combination maintenance page where you define new parts with a key flexfield for the part numbers.

  You would also have a code-combination reference page where you enter orders for parts, using the key flexfield to indicate what parts are included in the order. The page might also contain a code-combination filter, which you use to determine the acceptable values of your part numbers. This code-combination filter references the same key flexfield as the code-combination maintenance page and the code-combination reference page.

  The order of key flexfield segments in the application user interface corresponds to the order in which they were defined in the key flexfield metadata. You cannot reliably change that order at runtime. The user interface dynamically reads the displayed attributes from the view object and displays them in the same order that they occur in the view object. There are no attribute UI hints that you can use to override this behavior.

  Reordering key flexfield segments is not supported and can potentially create data integrity issues for code combinations, which are sequence-aware. Because of this, it is important that you plan the segment order of your key flexfields in advance.
The tasks to employ a key flexfield on a page include:

- Adding the key flexfield UI component to a form or a table
- Defining a default value for every SIN attribute to prevent application errors when a new row that contains key flexfield columns is added on an application page
- For ADF Form pages, adding code to ensure proper updating of reference mode and secondary mode SIN values
- For secondary-usage flexfield segments or for segments on a code-combination maintenance page, where the flexfield is in an ADF Table that is wrapped in an Applications Table component, adding functionality that dynamically refreshes the segment columns whenever the Applications Table component is refreshed by another component, such as a button or a search query

If you want changes in your key flexfield to trigger a partial update of another component, set the AutoSubmit UI property of the flexfield to True, and add the key flexfield ID to the PartialTriggers UI property of the other component.

**Caution:**
To ensure that the trigger works, you must append "CS" to the key flexfield ID. For example, if you want changes in the MyKeyFlex01 flexfield to trigger an update in another component, add "MyKeyFlex01CS" to that component's PartialTriggers property.

For more information about setting user interface properties, see Configuring Flexfield-Level User Interface Properties.

### 25.4.1.1 Add Key Flexfield UI Components to a Form or a Table

To incorporate a key flexfield into a UI form or table, you add the master view object to the page as a form or a table, and you drop the key flexfield view object onto the form or table.

Note that when the page creates a new row for the master view object, the value of the primary key of the row that references the key flexfield must be generated automatically, and this value cannot be changed.

**To add a key flexfield UI component to a form or a table:**

1. Create the user interface for the master view object:
   - In the Data Controls panel, select the master view object and drag it onto the page.
2. When prompted, select the type of user interface that you want to create:
   - ADF Form
   - Applications > Panel
   - ADF Table
   - Applications > Table
For an ADF Table, select the **Row Selection** option in the Edit Table Columns dialog.

3. In the Data Controls panel, select the key flexfield view object and drag it onto the page. Drop the flexfield view object onto a form or a table, and select the appropriate flexfield UI component. Figure 25-10 shows a key flexfield being dropped onto a form.

![Key Flexfield Dropped Onto a Form](image)

4. Using either sequence generation or default values, ensure that when the end user adds a row to the page, a valid value is generated for the master view object's primary key before the row is created. Also ensure that the primary key cannot be entered or edited by the user.

   **Caution:**

   You cannot use the code combination ID as part of the generated primary key.

5. If you are creating an editable table, select the table in the Structure window, expand the **Behavior** section of the Property Inspector and set the **EditingMode** attribute. If you want all the rows to be editable, then select **editAll**. If you want to enable the end user to click a row to make it editable, then select **clickToEdit**.

   If you select **clickToEdit**, then the editable row displays the concatenated flexfield segment values in an input text component. The end user can click an icon that is next to the editable flexfield to open a dialog box that has an input field for each segment. The flexfield values in the non-editable rows are displayed as read-only values. The user can click the icon that is next to a read-only flexfield value to open a window that displays the segment labels, values, and descriptions.

25.4.1.2 Ensure Proper Handling of New Rows

When a new row that contains key flexfield columns is added on an application page, every structure instance number (SIN) attribute must contain a valid value, so that the key flexfield user interface can be rendered with appropriate structures. Without default structures in reference mode, end users will not be able to select key
combinations for the new row. Without default structures in secondary mode, users will not be able to select segment values for the new row.

You can prevent application errors by defining a default value for each SIN attribute. Edit the foreign key entity object SIN attribute or the foreign key view object SIN attribute, and do one of the following:

- If the SIN value is static, set the **Value Type** to **Literal**, and specify the static value as the default.
- If the SIN value is dynamic, set the **Value Type** to **Expression**, and enter a Groovy expression to retrieve the appropriate SIN value and set it as the default.

**Note:**

For secondary-usage flexfield segments in table components, you can use the `defaultSIN` attribute in the JSPX file to define the default SIN value for situations where no rows exist.

For an Applications Table component in reference mode, the default SIN and structure for a new or modified row is just a starting point. You can always allow for runtime selection of a new structure by LOV or input field.

**Caution:**

For key flexfield secondary usages, the SIN value of the first row of a user interface table that contains secondary mode columns determines the secondary mode column structure to be used for the table. For any additional row that contains a different SIN value, the secondary mode columns that are not also part of the first row's structure will not render in the table. If there are no rows, then the value of the `defaultSIN` tag attribute from the JSPX file, if set, determines the secondary mode column structure.

For an Applications Table component, if the end user deletes all rows from the table, then your application can again set a new default SIN value for the first new row, and the secondary mode column structure corresponding to that SIN will be the valid structure for the table.

For an ADF Table, the secondary mode column structure (determined by the initial SIN value) cannot be changed after the table has been created, even if all rows are deleted.

For more information about setting attribute defaults, see the discussion about defining default values in the Setting Attribute Properties section of the Developing Fusion Web Applications with Oracle Application Development Framework.

### 25.4.1.3 Ensure Proper Updating of Reference Mode SIN values in an ADF Form or ADF Applications Table

It is best to not display the CCID or, if it is displayed, to make it a read-only field. When the end user clicks the popup icon, the popup requires the user to select a code combination whenever the SIN is changed for a row, and the CCID is set based on the user's selection.
If the CCID is not displayed in the user interface or if it is displayed as a read-only value, then you must ensure that the CCID for the current row in the view object is set to null whenever the SIN is changed by the end user.

If the form does display the CCID in an editable mode, then the application must force the end user to set the CCID to null, change the SIN value, and then enter the CCID value to update the concatenated value.

25.4.1.4 Ensure Proper Updating of Secondary Mode SIN Values in an ADF Form

When an end user makes changes to an existing ADF Form row that contains key flexfield secondary usage attributes, it might result in the need for that row to use a different structure instance. The row’s underlying view object retains the old SIN value, which produces a mismatch with the data and generates runtime errors. Your application must change the SIN value in the row so it uses the new structure instance. You add code to your page to ensure that this happens, as shown in Example 25-7.

Example 25-7   Code for Updating Modified SIN Values

```java
// Add the target for the component that contains
// the key flexfield secondary usage, which, in this case
// is a panelFormLayout component
AdfFacesContext.getCurrentInstance().addPartialTarget(pfl1);

// Get the handle of the child iterator binding. This is the same iterator
// that you get when you drag the key flexfield secondary usage onto a JSPX page
DCIteratorBinding childBinding =
    bindingControl.findIteratorBinding("Kff1PaInstanceIterator");

// Get the view object from the child iterator
ViewObject childVO = childBinding.getViewObject();

// Get the current row from this view object
ViewRowImpl childRow = (ViewRowImpl) childVO.getCurrentRow();

// Update the SIN in the child view object
KFFPViewDefImpl childViewDef = (KFFPViewDefImpl)childRow.getViewDef();
childRow.setAttribute((childViewDef.getStructureInstanceNumberAttribute()).getName(), sinValue);
```

Caution:

You cannot use this solution in an ADF Table or an Applications Table component. Dynamically changing the SIN at runtime is supported only for an ADF Form.

25.4.1.5 Dynamically Refresh Segments on a Code-Combination Maintenance Page or Secondary Usage Segments

If you have segments on a code-combination maintenance page or you have secondary usage segments, and those segments are in an ADF Table that is wrapped in an Applications Table component that is refreshed by another component, such as a button or a search query, then you must add functionality to dynamically refresh the segment columns.

To refresh the flexfield segments based on the current iterator rowset data, create a listener handler method in the flexfield’s backing bean and bind the listener to the
component that is initiating the table refresh. The listener must first call the default
listener and then call
DescriptiveFlexfield.updateFlexColumns(RichTable), where RichTable
is the binding for the table that contains the flexfield.

Example 25-8 shows an example of a custom flexfield handler for a query event. The
method first calls invokeMethodExpression to call the original query listener, and
then calls updateFlexColumns with the table component that contains the flexfield
as the parameter. Example 25-9 shows the binding of the custom flexfield handler to
the query component.

Example 25-8   Flexfield Listener

public void customKffSearchQueryListener(QueryEvent queryEvent)
{
    invokeMethodExpression(
        "#{bindings.KffCriteriaQuery.processQuery}",
        Object.class,QueryEvent.class,queryEvent);
    DescriptiveFlexfield.updateFlexColumns(appTable);
}

Example 25-9   Binding the Flexfield Listener to the Search Query

<af:query id="qryId1"
    headerText="#{applcoreBundle.QUERY_SEARCH_HEADER_TEXT}"
    disclosed="true"
    value="#{bindings.criteriaQuery.queryDescriptor}"
    model="#{bindings.criteriaQuery.queryModel}"
    queryListener="#{backingBeanScope.dffBean.customKffSearchQueryListener}"
    queryOperationListener="#{bindings.KffCriteriaQuery.processQueryOperation}"
    resultComponentId="::AT2:_ATp:ATt2"/>

Note:

You do not need to handle flexfield refreshing for standard Applications Table
create and delete operations. However, custom create and delete operations
must handle the refreshing of flexfields.

25.4.1.6 What Happens When You Add a Key Flexfield to a Page

A key flexfield is implemented in the user interface as a code combination LOV rather
than as individual segments on the page. You can enter a combination code directly
into the code combination LOV input.

Figure 25-11 shows an example of a key flexfield used in a form on an application
page.

Figure 25-11   Example of a Key Flexfield In a Form

<table>
<thead>
<tr>
<th>Primary Key Code</th>
<th>VS_IND_COE_DEP_CHR_ON_CHR:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sn</td>
<td>8</td>
</tr>
<tr>
<td>Cdd</td>
<td>3</td>
</tr>
<tr>
<td>Key Flexfield</td>
<td>CA.Redwood City</td>
</tr>
</tbody>
</table>

Employ Key Flexfield UI Components on a Page
Figure 25-12 shows an example of a key flexfield used in a table on an application page.

**Figure 25-12  Example of a Key Flexfield in a Table**

<table>
<thead>
<tr>
<th>PartNumber</th>
<th>Description</th>
<th>UnitCost</th>
<th>Bin</th>
<th>Account</th>
<th>CMU Demo Parts Flexfield</th>
<th>Inspection Required</th>
<th>Inspection Date</th>
<th>Lead Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>3729662</td>
<td>Plastic Wheels</td>
<td>2.25</td>
<td>2209</td>
<td>AB10000</td>
<td>Y</td>
<td>1/1/09</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3722009</td>
<td>Padded Seat</td>
<td>39.99</td>
<td>12201</td>
<td>CMU_DWGO_CPLA</td>
<td>DEV.AB10.000</td>
<td>N</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3722791</td>
<td>Executive Seat Back</td>
<td>99.99</td>
<td>12201</td>
<td>CC_ACCT_OCG_DIVY</td>
<td>DEV.AB10.000.000.0000</td>
<td>Y</td>
<td>1/2/09</td>
<td>2</td>
</tr>
<tr>
<td>3722854</td>
<td>Standard Seat Back</td>
<td>29.99</td>
<td>12201</td>
<td>CPOQ_OC_CC_ACCT_DIVY</td>
<td>DEV.AB10.000.000.0000</td>
<td>Y</td>
<td>1/3/09</td>
<td>3</td>
</tr>
<tr>
<td>3723027</td>
<td>Chair Base</td>
<td>15.25</td>
<td>12201</td>
<td>CPOQ_CC_ACCT_DIVY</td>
<td>SALES.000.000.000.000.000.0000</td>
<td>N</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>3723920</td>
<td>3 Powercomputer Case</td>
<td>99.99</td>
<td>12201</td>
<td>CC_ACCT_OCG_DIVY</td>
<td>CMU.000.000.000.000.0000</td>
<td>N</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3723879</td>
<td>St. Speakercomputer</td>
<td>86.99</td>
<td>12201</td>
<td>CC_ACCT_OCG_DIVY</td>
<td>DEV.AB10.000.000.0000</td>
<td>N</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>3724468</td>
<td>1GB Memory Chip</td>
<td>119.00</td>
<td>12201</td>
<td>CC_ACCT_OCG_DIVY</td>
<td>DEV.AB10.000.000.0000</td>
<td>N</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

**Caution:**

When your flexfield is in a table that is displayed within a popup, and the table's *contentDelivery* attribute is set to immediate, you must also set the popup's *contentDelivery* attribute to immediate to ensure that the key flexfield UI component is rendered in the table. For any other value of the popup's *contentDelivery* attribute, the flexfield column in the table will be blank.

For more information about tables and popups, see the "Using Tables, Trees, and Other Collection-Based Components" and "Using Popup Dialogs, Menus, and Windows" chapters of the *Developing Web User Interfaces with Oracle ADF Faces*.

Key flexfield segments always appear as form fields or table columns in the same order that their corresponding attributes appear in the underlying view object.

In screenreader mode, a labeled icon of three horizontal bars appears next to the key flexfield input text field. When the end user clicks the icon, instead of the standard popup, a page is displayed that shows the segment details. The user clicks **Done** to return to the prior page.

For key flexfields in forms and in tables, you can click the search icon to select a valid new flexfield code combination using individual segment values as criteria, as shown in Figure 25-13.
25.4.2 How to Incorporate Key Flexfields into a Query Search Form

In reference mode, you can include key flexfield view object attributes as search criteria in an advanced mode query search form. This form enables end users to define extemporaneously the criteria to search for metadata in the view object for the reference table, which is the product table that has a foreign key reference to the combinations table, and its linked key flexfield view object. Users can select which attributes of the key flexfield view object to use as search criteria.

To incorporate key flexfields into a query search form:

1. Set up the business component model layer.
2. Create the query search form.

25.4.2.1 Set Up the Business Component Model Layer

To set up the business component model layer for the search form, you define the view criteria in the view object for the reference table, generate the row implementation class for the reference view object, and override the getCriteriaItemClause() method in that row implementation class.

For more information about defining view criteria, see the Defining SQL Queries Using View Objects chapter of the Developing Fusion Web Applications with Oracle Application Development Framework.

Before you begin:

1. Create a view link between the view object for the reference table and the key flexfield polymorphic view object as described in How to Create Key Flexfield View Links.
2. Add the key flexfield’s view instance to the product application module as described in How to Add an Instance of a Key Flexfield View Object to the Product Application Module.

3. Nest the key flexfield application module instance in the product application module, as described in How to Nest an Instance of the Key Flexfield Application Module in the Product Application Module.

4. Ensure that the discriminator attribute in the view object for the reference table, such as the SIN attribute, is enabled to display a list of values. You can find this information on the Attributes navigator tab for the view object. Also, ensure that the Auto Submit property for the discriminator attribute is set to true in Control Hints.

For information about enabling a list of values for an attribute and setting control hints, see the Defining SQL Queries Using View Objects chapter of the Developing Fusion Web Applications with Oracle Application Development Framework.

To set up the business component model layer:

1. In the Application Navigator, double-click the view object for the reference table.

2. In the overview editor, click the Query navigation tab.

3. In the View Criteria section, click the Add icon.

4. In the Create View Criteria dialog, enter the name of the view criteria to identify its usage in your application.

5. Click Add Group, and click Add Item.

6. Select the discriminator, such as Sin, from the Attribute dropdown list.

7. Accept the default values of Equal to for the Operator and Literal for the Operand.

8. Select the Group node that you just added, and click Add Item.

9. From the Attribute dropdown list, select the view accessor for the view link between your application’s reference view object and the key flexfield view object.

10. Accept the default Exists value for Operator and Inline View Criteria for Operand.

11. Select the bottom node in the View Criteria tree as shown in Figure 25-14.

Figure 25-14  Bottom Node in View Criteria Tree
12. Select the attribute that corresponds to the discriminator, such as _STRUCTURE_INSTANCE_NUMBER, from the Attribute dropdown list.

13. Accept the default values of Equal to for the Operator and Literal for the Operand.

14. Click OK.

15. Click the Source tab and locate the view criteria that you added in Step 3. Within this view criteria, identify the nested view criteria that contains the <ViewCriteriaItem> element for _STRUCTURE_INSTANCE_NUMBER.

16. In the <CustomProperties> element, add a <Property> element. Set the Property Name to FND_ACFF_IsQueryNestedCriteria and set the Value to true as shown in bold in Example 25-10.

17. Save your changes.

Example 25-10 Property FND_ACFF_IsQueryNestedCriteria in the Nested Criteria

```xml
<ViewCriteria
    Name="AcctKffVONestedCriteria"
    ViewObjectName="oracle.apps.fnd.applcore.crmdemo.flex.acct.view.AcctKffVO"
    Conjunction="AND">
    <ViewCriteriaRow Name="vcrow487" UpperColumns="1">
        <ViewCriteriaItem
            Name="_STRUCTURE_INSTANCE_NUMBER"
            ViewAttribute="_STRUCTURE_INSTANCE_NUMBER"
            Operator="="
            Conjunction="AND"
            Required="Optional"/>
    </ViewCriteriaRow>
    <Properties>
        <CustomProperties>
            <Property Name="FND_ACFF_IsQueryNestedCriteria" Value="true"/>
        </CustomProperties>
    </Properties>
</ViewCriteria>
```

25.4.2.2 Create the Query Search Form

To create a query search form that contains a key flexfield, add an ADF Query Panel with Table component to the page and drop the key flexfield into the table. Then, create a custom bean and attach the bean to the query.

For more information about working with search forms, see the "Creating ADF Databound Search Forms" chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

Before you begin:

1. Prepare the business model component as described in Setting Up the Business Component Model Layer.

2. If you are working with secondary-usage flexfield segments and the flexfield is an ADF Table that is wrapped in an Applications Table component, review Ensuring Proper Handling of New Rows and Dynamically Refreshing Segments on a Code-Combination Maintenance Page or Secondary Usage Segments.
To create the query search form:

1. Open the JSPX page to which you want to add the search form.

2. From the Data Controls panel, select the reference view object’s data collection and expand the Named Criteria node to display a list of named view criteria.

3. Drag the view criteria that you created in Setting Up the Business Component Model Layer and drop it onto the page or onto the Structure window.

4. From the context menu, choose Query > ADF Query Panel with Table, as shown in Figure 25-15.

Figure 25-15 Query Context Menu

5. If you are working with a secondary (partial) usage or you are working with a maintenance page, perform the following substeps to enable the execution of a saved search when the page is loaded for the first time:

   a. From the Property Inspector for the table in which the flexfield is displayed, obtain the TableId value.

   b. In the Property Inspector for the Query component, set the value of the ResultComponentId property to the TableId value.

6. In the Edit Table Columns dialog, you can rearrange any column and select table options.

7. In the Data Controls panel, select the key flexfield view object, drop it into the table, and choose Create > Oracle Key Flexfield Column to add the key flexfield to the table, as shown in Figure 25-16.
8. In the user interface project, create a custom bean that implements the `oracle.adf.view.rich.event.QueryOperationListener` interface as shown in Example 25-11.

Set `VIEW_CRITERIA_NAME` to the name of the view criteria, and set `KFF_ACCESSOR_NAME` to the view accessor from the view link between the view object for the reference table and the key flexfield view object.

This custom bean will be triggered when a value is selected from the LOV component for the discriminator attribute, such as the LOV for the SIN attribute. When invoked, the `processQueryOperation()` method is called. The `JUFormBinding` that is associated with the view criteria is accessed to extract the view criteria.

The `applyDiscriminator()` method extracts the `ViewCriteriaItem` for the discriminator attribute, gets the value that was selected from the discriminator's LOV component, and loads into the query panel the key flexfield's subtypes with a matching discriminator value.

9. Complete the following steps to attach the custom bean to the query:

   a. Open the JSPX page.

   b. In the Structure window, select `af:query`.

   c. In the Property Inspector, expand the `Behavior` section.

   d. In the `QueryOperationListener` field, enter an EL expression that resolves to the custom bean's `processQueryOperation()` method, such as `#{CustomBean.processQueryOperation}`.

Example 25-11  Custom Listener Java Class

```java
package oracle.apps.fnd.applcore.flex.test.backing;

import java.util.List;
import javax.faces.event.AbortProcessingException;
import oracle.adf.model.BindingContext;
import oracle.adf.model.binding.DCBindingContainer;
import oracle.adf.view.rich.event.QueryOperationEvent;
import oracle.adf.view.rich.event.QueryOperationListener;
import oracle.adf.view.rich.model.AttributeCriterion;
import oracle.adf.view.rich.model.Criterion;
import oracle.adf.view.rich.model.QueryDescriptor;
```
import oracle.jbo.uicli.binding.JUFormBinding;
import oracle.jbo.ViewCriteria;
import oracle.jbo.ViewCriteriaItem;
import oracle.jbo.ViewCriteriaRow;
import oracle.jbo.common.ViewCriteriaItemImpl;
import oracle.jbo.uicli.binding.JUSearchBindingCustomizer;

public class CustomBean implements QueryOperationListener{
    public CustomBean() {
        super();
    }

    private static final String BINDING_SUFFIX="Query";
    private static String VIEW_CRITERIA_NAME="PartsKFFQueryCriteria";
    private static String KFF_ACCESSOR_NAME="PartsAcctKffKffVL";
    private static String KFF_DISCRIMINATOR_ATTR_NAME="_STRUCTURE_INSTANCE_NUMBER";
    private static String MASTER_VO_ATTR_FROM_WHICH_KFF_DISC_DERIVED="Sin";

    public void processQueryOperation(QueryOperationEvent queryOperationEvent)
            throws AbortProcessingException{
        QueryDescriptor descriptor = (QueryDescriptor) queryOperationEvent.getDescriptor();
        AttributeCriterion attr = queryOperationEvent.getAttributeCriterion();
        if (queryOperationEvent.getOperation()==
            QueryOperationEvent.Operation.CRITERION_UPDATE){
            BindingContext bcx= BindingContext.getCurrent();
            DCBindingContainer bc = (DCBindingContainer) bcx.getCurrentBindingsEntry();
            JUFormBinding bnd = (JUFormBinding)
            bc.findExecutableBinding(VIEW_CRITERIA_NAME+BINDING_SUFFIX);
            if(bnd!=null){
                String vcName = JUSearchBindingCustomizer.getCriteriaName(bnd);
                applyDiscriminator(bnd, vcName);
            }
        }
    }

    public void applyDiscriminator(JUFormBinding ctr, String vcName){
        ViewCriteria vc = JUSearchBindingCustomizer.getViewCriteria(ctr, vcName);
        ViewCriteriaRow r = (ViewCriteriaRow) vc.getCurrentRow();
        List<ViewCriteriaItem> criteriaItemList = r.getCriteriaItems();
        Object proxyval = null;
        for(ViewCriteriaItem item : criteriaItemList){
            if (item.getName().equals(MASTER_VO_ATTR_FROM_WHICH_KFF_DISC_DERIVED)){
                proxyval = item.getValue();
            }
            if (item instanceof ViewCriteriaItem){
                ViewCriteriaItemImpl itemimpl = (ViewCriteriaItemImpl) item;
                if(itemimpl.getName().equals(KFF_ACCESSOR_NAME)){
                    ViewCriteria nvc = itemimpl.getNestedViewCriteria();
                    ViewCriteriaRow nvcr = (ViewCriteriaRow) nvcr.getCurrentRow();
                }
            }
        }
    }
}
List<ViewCriteriaItem> ncriteriaItemsList = nvcr.getCriteriaItems();
for(ViewCriteriaItem nitem: ncriteriaItemsList){
    if (nitem.getName().equals(KFF_DISCRIMINATOR_ATTR_NAME))
        nitem.setValue(proxyval);
}

} //if instanceof

} //end for

} //end for

} //end for

25.4.3 How to Configure Key Flexfield UI Components

You can configure various aspects of a key flexfield UI component to customize the behavior of the flexfield as a whole, or on a segment-by-segment basis. You can control your key flexfield’s behavior in the application user interface by modifying properties at the flexfield level, at the segment label level, and at the secondary usage level.

25.4.3.1 Configure Flexfield-Level User Interface Properties

Right-click a key flexfield’s UI component on the page, then select Properties from the context menu to view and modify its properties in the Property Inspector, as shown in Figure 25-17.

Figure 25-17 Key Flexfield Property Inspector — Common Tab

The significant properties on the Common, Data, Style, Behavior, and Other property tabs are listed in Table 25-6.
### Table 25-6  Key Flexfield Properties

<table>
<thead>
<tr>
<th>Tab &gt; Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common &gt; Id</td>
<td>This property gives the ID of the flexfield.</td>
</tr>
<tr>
<td>Common &gt; Rendered</td>
<td>This property indicates whether the flexfield is rendered on the application page. Values can be true (default) or false. When this property is set to false, the flexfield is not sent to the client. Expression language (EL) expressions are allowed on ADF Form and ADF Table components. On ADF Table components, you use expressions to control this property on a row-by-row basis.</td>
</tr>
<tr>
<td>Common &gt; Label</td>
<td>This property is the prompt that should be rendered on the page. It is also used for the title of popup components. Note that if the Label property does not have a value, the default title of a popup component is Key Flexfield. Therefore, you must set this value to the name of the flexfield to ensure that popup components display the correct title, and not the default. The value should be applied from a resource bundle.</td>
</tr>
<tr>
<td>Common &gt; Value</td>
<td>This property is the value of the flexfield. This should be an EL expression pointing to an iterator object. This field is also visible on the Data tab.</td>
</tr>
<tr>
<td>Data &gt; Accessor</td>
<td>This property is the name of the accessor between the (consumer) product view object and the flexfield view object.</td>
</tr>
<tr>
<td>Style &gt; StyleClass</td>
<td>This property is the style class of the flexfield. A style class allows you to group a set of inline styles.</td>
</tr>
<tr>
<td>Style &gt; InlineStyle</td>
<td>This property is the inline style of the component. This is a string of CSS styles that can set individual properties such as background color, font style, or padding.</td>
</tr>
<tr>
<td>Style &gt; Width</td>
<td>This property is the width in characters of the text field in which the key flexfield value is displayed on the page. This value is 30 by default.</td>
</tr>
<tr>
<td>Behavior &gt; Required</td>
<td>This property indicates whether the key flexfield must have a value. Values of this property can be true or false (default). When this property is set to true, the page containing this key flexfield cannot be submitted unless the flexfield has a value. EL expressions are allowed on ADF Form and ADF Table components. On ADF Table components, you use expressions to control this property on a row-by-row basis.</td>
</tr>
</tbody>
</table>
### Table 25-6 (Cont.) Key Flexfield Properties

<table>
<thead>
<tr>
<th>Tab &gt; Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Behavior &gt; ReadOnly</td>
<td>This property indicates whether the key flexfield is rendered as read-only. Values can be <code>true</code> or <code>false</code> (default). When this property is set to <code>true</code>, the flexfield segment values are rendered, but they cannot be modified, and the associated popup or LOV lookup controls do not appear. Instead, an icon is displayed. When the mouse hovers over the icon, a window appears that displays the segment labels, values, and descriptions. This behavior overrides the <code>Disabled</code> property. EL expressions are allowed on ADF Form and ADF Table components. On ADF Table components, you use expressions to control this property on a row-by-row basis.</td>
</tr>
<tr>
<td>Behavior &gt; Disabled</td>
<td>This property indicates whether the UI control associated with this key flexfield can be operated. Values can be <code>true</code> or <code>false</code> (default). When this property is set to <code>true</code>, the flexfield segment values and the associated popup or LOV lookup controls are rendered, but are dimmed and cannot be modified or operated. The <code>ReadOnly</code> property, when set to <code>true</code>, takes precedence over this property. EL expressions are allowed on ADF Form and ADF Table components. On ADF Table components, you use expressions to control this property on a row-by-row basis. Note that the flexfield will be disabled if the current master row for the flexfield does not have a valid SIN value defined.</td>
</tr>
<tr>
<td>Behavior &gt; PartialTriggers</td>
<td>This property contains the IDs of the components that should trigger a partial update in the flexfield (<code>String[]</code>). EL expressions are allowed.</td>
</tr>
<tr>
<td>Behavior &gt; ValueChangeListener</td>
<td>This property is a method reference to a value change listener (<code>javax.faces.el.MethodBinding</code>). It requires an EL expression. The value change listener takes effect if the value of the key flexfield is changed either manually in the key flexfield text field, or by using the key flexfield LOV popup.</td>
</tr>
<tr>
<td>Behavior &gt; Binding</td>
<td>This property is an EL reference that will store the component instance on a bean (<code>oracle.apps.fnd.appcore.flex.ui.KFFComp</code>). This property requires an EL expression.</td>
</tr>
</tbody>
</table>
### Table 25-6 (Cont.) Key Flexfield Properties

<table>
<thead>
<tr>
<th>Tab &gt; Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other &gt; <strong>AutoSubmit</strong></td>
<td>This property indicates whether key flexfield values that are entered by the end user should automatically be submitted directly upon entry. Values can be true or false (default). When this property is set to true, and the end user changes the value of the flexfield in the key flexfield text field or by using the key flexfield LOV popup, the new value will be submitted to the product view object immediately. When this property is set to false, the new value will be submitted only when the entire page is submitted.(^1) EL expressions are allowed on ADF Form and ADF Table components. On ADF Table components, you use expressions to control this property on a row-by-row basis.</td>
</tr>
<tr>
<td>Other &gt; <strong>DefaultSIN</strong></td>
<td>This property is for secondary usage only. It defines the default SIN value to use to define the structure when no rows exist.</td>
</tr>
<tr>
<td>Other &gt; <strong>Changed</strong></td>
<td>This property indicates whether the changed indicator icon is displayed on the component. Values can be true or false (default). When this property is set to True, the changed indicator icon is displayed. EL expressions are allowed on ADF Form and ADF Tables. On ADF Table components, you use expressions to control this property on a row-by-row basis.</td>
</tr>
<tr>
<td>Other &gt; <strong>Simple</strong></td>
<td>This property indicates whether the key flexfield's label should be hidden. Values of this property can be true or false (default). When this property is set to true, the label is hidden. Note that if the Simple property is set to true and the flexfield is placed inside a PanelLabelAndMessage component, the flexfield might not align properly with the other components in the PanelLabelAndMessage component. EL expressions are allowed on ADF Form and ADF Table components. On ADF Table components, you use expressions to control this property on a row-by-row basis.</td>
</tr>
<tr>
<td>Other &gt; <strong>Visible</strong></td>
<td>This property indicates whether the key flexfield appears on the page. Values can be true (default) or false. When this property is set to false, the key flexfield is sent to the client but the client does not display it. EL expressions are allowed on ADF Form and ADF Table components. On ADF Table components, you use expressions to control this property on a row-by-row basis.</td>
</tr>
</tbody>
</table>

\(^1\) If you want changes in your key flexfield to trigger a partial update of another component, set the **AutoSubmit UI** property of the flexfield to true, and add the key flexfield ID to the **PartialTriggers UI**
property of the other component. To ensure that the trigger works, you must append "CS" to the key flexfield ID. For example, if you want changes in the `MyKeyFlex01` flexfield to trigger a partial update in another component, add "MyKeyFlex01CS" to that component's `PartialTriggers` property.

**Note:**
The *Behavior > Mode* property defines the user interface mode of the key flexfield component.

Only the single default value is supported, which renders the key flexfield as a single LOV.

### 25.4.3.2 Configure Label-Based Segment UI Properties

Key flexfields support finer control of segments in the user interface based on their segment labels, using several additional properties that you can set in the flexfield XML with literal values or EL expressions. These properties are attributes of the `flexfieldLabeledSegmentHint` component.

**Note:**
This component can be used only to configure the segment UI properties of key flexfield secondary usages, and only if a segment label is applied.

The following properties can be used to define usage-specific behavior for one or more key flexfield segments, identified by segment label. These property settings apply to all segments that have the specified segment label assigned.

- **SegmentLabel**: This string property specifies the segment label of the segment being configured. This string property is required.
- **Rendered**: This boolean property indicates whether the segment is visible on the application page.
- **Required**: This boolean property indicates whether the segment must have a value.
- **Readonly**: This boolean property indicates whether end users can modify the segment.
- **Label**: This string property provides a display label for the UI component.
- **ShortDesc**: This string property provides a short description of the UI component. This text is commonly used by user agents, such as browsers, to display tooltip help text, in which case the behavior for the tooltip is controlled by the user agent.
- **Columns**: This integer specifies the width of the text control, in terms of the number of characters shown. The number of columns is estimated based on the default font size of the browser.
Note:

If you set a segment’s Required property to true in the flexfield metadata, for validation purposes you cannot override this by resetting it to false in the page metadata. You can, however, do the reverse: Change a nonrequired segment to be required in the page metadata.

The Label, ShortDesc and Columns properties are expected to apply to a single segment, so it is best to use them when only one segment has this segment label assigned.

The default values of these properties are derived from the flexfield metadata, but you can override the default values at the page level.

For information about using EL expressions, see the Creating ADF Data Binding EL Expressions section in the Developing Fusion Web Applications with Oracle Application Development Framework.

You apply these properties by dragging and dropping the following component from the Component Palette into the key flexfield, as a child of the keyFlexfieldPartial component:

```html
<fnd:flexfieldLabeledSegmentHint propertyname1="value"
[propertyname2="value" [propertyname3="value"
[propertyname4="value" [propertyname5="value"
[propertyname6="value" [propertyname7="value"]]]]]]>```

### 25.4.3.3 Configure Secondary Usage UI Properties

Key flexfields support finer control of secondary usages in the user interface with several additional properties that you can set with literal values or EL expressions. These properties are attributes of the keyFlexfieldPartial component. By using EL expressions at the page level, you can programmatically override the key flexfield metadata at runtime.

For example, Oracle Assets has a single page that is used for both the Create Asset and Update Asset activities. When creating an asset, the Asset Category key flexfield on this page should be updatable; when updating an asset, the flexfield should be read-only. This setting can be programmatically managed using the readOnly property based on a page parameter that indicates whether the page is in Create mode or Update mode.

The following boolean properties can be used to specify usage-specific behavior for the entire key flexfield secondary usage:

- **rendered**: Indicates whether the flexfield is visible on the application page.
- **required**: Indicates whether the flexfield must have a value.
- **readOnly**: Indicates whether end users can modify the flexfield.

The default values of these properties are derived from the flexfield metadata, but you can override them at the page level.
Note:

If you set a segment’s `required` property to `true` in the flexfield metadata, for validation purposes you cannot override this by resetting it to `false` in the page metadata. You can, however, do the reverse: Change a non-required segment to be required in the page metadata.

For information about using EL expressions, see the "Creating ADF Data Binding EL Expressions" section in the Developing Fusion Web Applications with Oracle Application Development Framework.

You apply these properties by dragging and dropping the following component from the Component Palette into the key flexfield element:

```xml
<fnd:KeyFlexfieldPartial propertyname1="value"
[propertyname2="value" [propertyname3="value"]]>  
```
This chapter discusses the development activities for taking advantage of key flexfield partial usages, code-combination filters, and other advanced features. This chapter includes the following sections:

- Introduction to Advanced Features of Key Flexfields
- Using Key Flexfield Advanced Features in Reference Mode
- Completing the Development Tasks for Key Flexfields in Secondary Mode
- Working with Code-Combination Filters for Key Flexfields

26.1 Introduction to Advanced Features of Key Flexfields

Key Flexfields have advanced features like code combination filters and reference mode. The criteria for filtering the list of code combinations that can be referenced in a given combinations table are known as code combination constraints. Combination constraints are view object properties and are not applied on any entity objects. The type of usage when there is no direct relationship between the product table and the key flexfields combinations table is known as secondary storage.

26.2 Use Key Flexfield Advanced Features in Reference Mode

Key flexfield advanced features include code combination constraints, programmatic access to segment labels, making key flexfields available for use in Oracle Business Intelligence, and working with flexfields from a worksheet using ADF Desktop Integration.

26.2.1 How to Define Code Combination Constraints

Code combination constraints are criteria for filtering the list of code combinations that can be referenced in a given combinations table. While the set of code combinations in the table is not changed, each reference table (product table with foreign key references to these code combinations) can have its own associated code combination constraints.

For example, the key flexfield MTL_SYSTEM_ITEMS has a Purchasable flag, which can be set to the value Y or N. You can implement an extra WHERE clause on the Oracle Purchasing product view object that enables Order Management to restrict the displayed items to only those with Purchasable set to Y.

Code combination constraints are applied in the following situations:
• When an end user opens a key flexfield popup window to search for a code combination.

• When the code-combination ID attribute of a view object for a reference table is set programmatically.

Code combination constraints are not applied when an existing foreign key reference to a code combination is resolved into individual segments (or a concatenated string) for display.

Combination constraints are view object properties and are not applied on any entity objects.

You create a view accessor to define a code combination constraint. You can define the following types of code combination constraints:

• Extra WHERE clause
• Validation date
• Validation rules
• Dynamic combination creation allowed

26.2.1.1 Create a View Accessor to Define a Code Combination Constraint

In a key flexfield ADF Business Components model, the code combination constraints are defined in a view object that references the code combinations (the reference view object). Although these constraints are, in a way, validation rules for code combinations, they are not ADF Business Components validators.

You define code combination constraints as bind parameter values in a view accessor. The name of this view accessor is derived from the name of the view link accessor to the key flexfield view object for which you want to constrain code combinations.

To define a code combination constraint:

1. Open the foreign key view object that references the code combinations.

2. Click the Add icon in the View Accessors section to display the View Accessors dialog.

3. Create a view accessor to the key flexfield’s base view object, as shown in Figure 26-1.
The destination of the view accessor is the key flexfield's base view object. The name of the view accessor must be derived from the name of the view link accessor to the key flexfield view object, and must take the following form:

`viewlinkaccessornameConstraints`

For example, for a view link accessor named `AcctKff`, Figure 26-1 shows the accessor name `AcctKffConstraints`.

4. Edit the view accessor to define the bind parameter values, as shown in Figure 26-2.

**Note:**

You do not need to select any view criteria for this activity. Only the bind parameter values are needed to define a code combination constraint.

If you do not see any bind parameters, it is likely that you have just re-created the business components and overwritten the old ones. You can close the application and remove it from Oracle JDeveloper. When you open the application again, JDeveloper should load the latest definitions, including the bind parameters.
There are four types of code combination constraints. To apply a constraint type, provide a value for the appropriate bind parameter for the constraint type as shown in the following list. If no value is provided, that constraint type is not enabled.

- **Extra WHERE clause**: This constraint is invoked with the bind parameter Bind_ExtraWhereClause. You can also incorporate the predefined bind parameters BindVar0 through BindVar9.
  
  For information about how to provide a value for this bind parameter, see Constraining Code Combinations by an Extra WHERE Clause.

- **Validation date**: This constraint is invoked with the bind parameter Bind_ValidationDate.
  
  For information about how to provide a value for this bind parameter, see Constraining Code Combinations by Validation Date.

- **Value attribute validation rules**: This constraint is invoked with the bind parameter Bind_ValidationRules.
  
  For information about how to provide a value for this bind parameter, see Constraining Code Combinations by Validation Rules.

- **Dynamic combination creation allowed**: This constraint is invoked with the bind parameter Bind_DynamicCombinationCreationAllowed.
  
  For information about how to provide a value for this bind parameter, see Enabling or Disabling Dynamic Combination Creation for a Specific Usage.

Edit only the bind parameters that you need, and leave the others blank. You can use Groovy expressions as bind-parameter values. This means that the constraints can come indirectly from a view attribute, the view object, or a Java method.
Note:
You can ignore the Row-level bind values exist option, because the frequency of evaluation of bind parameters is predetermined, as follows:

- View object level (evaluated once)
  - Bind_ExtraWhereClause
  - Bind_ValidationRules
- Row level (evaluated every time)
  - BindVar0 through BindVar9
  - Bind_ValidationDate
  - Bind_DynamicCombinationCreationAllowed

26.2.1.2 Constrain Code Combinations by an Extra WHERE Clause
You can use the view accessor’s Bind_ExtraWhereClause parameter to filter the list of code combinations that can be referenced in a given combinations table. The extra WHERE clause is appended to the existing WHERE clause of the key flexfield view object.

To set the Bind_ExtraWhereClause parameter:

1. Create the view accessor and open it for editing as described in How to Define Code Combination Constraints.

2. In the Edit View Accessor dialog, set the Bind_ExtraWhereClause value.

   The extra WHERE clause can use bind parameters. The value of Bind_ExtraWhereClause should be a SQL fragment that may contain references to columns of the combinations table, or the predefined bind parameters.

   To refer to the combinations table, use ${COMBINATION_TABLE}; for example, ${COMBINATION_TABLE}.MY_COLUMN.

   To refer to one of the 10 predefined bind parameters, BindVar0 through BindVar9, use a colon and the bind parameter name; for example, :BindVar3.

   The following is an example of an extra WHERE clause code combination constraint as a SQL expression:

   (:BindVar0 IS NULL) OR (${COMBINATION_TABLE}.MY_COLUMN = :BindVar0)

   You can also express this as a Groovy string constant. Be sure to escape the dollar sign with a backslash:

   "(:BindVar0 IS NULL) OR (\${COMBINATION_TABLE}.MY_COLUMN = :BindVar0)"

26.2.1.3 Constrain Code Combinations by Validation Date
You can use the view accessor’s Bind_ValidationDate parameter to filter the list of code combinations that can be referenced in a given combinations table. If you provide a value for Bind_ValidationDate, then this validation date is used instead of the current database date when searching for a code combination. The code combinations returned are those that are active on the specified date.
If a code combination's `start_date_active` attribute is NULL, it is considered to have always been active in the past, up to its `end_date_active` date. If a code combination's `end_date_active` attribute is NULL, it is considered to be active starting from its `start_date_active` date indefinitely into the future.

**Note:**

Note that a date constraint is always required when searching for a code combination. If you do not supply a validation date, the current database date will be used.

---

**To set the Bind_ValidationDate parameter:**

1. Create the view accessor and open it for editing as described in [How to Define Code Combination Constraints](#).

2. In the Edit View Accessor dialog, set the `Bind_ValidationDate` value.

   The value of `Bind_ValidationDate` should be a normalized `java.sql.Date` object; that is, the hour, minute, second, and millisecond should be set to zero. You can use the method `oracle.apps.fnd.applcore.oaext.model.OAUtility#getSQLDate` to normalize the date.

   One way to construct a normalized date for testing purposes is to use `java.sql.Date.valueOf(String s)` with the date as a literal string in the form `yyyyMMdd`. In a search user interface, the supplied validation date also affects the list of values of a segment. For example, the end user may pick a value for a segment from a list of values, then use the segment value to search for a code combination. The list of values of the segment will be constrained by the supplied validation date.

---

### 26.2.1.4 Constrain Code Combinations by Validation Rules

You use validation rules to constrain code combinations. The validation rules for a given key flexfield are authored by the product team that owns the flexfield. They are valid only for use as code combination constraints, and should not be confused with other types of validation rules. Validation rules are stored in the flexfield metadata table `FND_KF_VRULES` and are delivered along with the key flexfield definition. The rule authors are your best source of information about the applicability of the validation rules for a flexfield, and the rule codes you should use to reference them.

You can use the view accessor's `Bind_ValidationRules` parameter to filter the list of code combinations that can be referenced in a given combinations table. If you provide a value for `Bind_ValidationRules`, the validation rules are translated into a SQL fragment, and the SQL fragment is appended to the `WHERE` clause of the key flexfield view object.

**Note:**

Because key flexfield secondary usages do not include a code combination, and validation rule constraints currently apply only to code combinations, they do not apply in the case of key flexfield secondary usages.
26.2.1.4.1 How to Create Validation Rules

Use the `create_vrule(...)` procedure from the `FND_FLEX_KF_SETUP_APIS` PL/SQL package to register a flexfield segment's validation rule. Validation rules apply only to segments that are validated against a list-validated value set. If the segment is validated against a format-only value set, the validation rules are ignored.

Note that when a segment is labeled with multiple segment labels, its validation rules are joined with an `AND` operator in the `WHERE` clause.

When the `ALWAYS_APPLIED_FLAG` is set to `Y`, the validation rule is always applied, such as when a combination is validated by C or PL/SQL validation APIs or a combination is validated by a business component. When the `ALWAYS_APPLIED_FLAG` is set to `N`, the validation rule is applied only when the rule is included in the list of validation rules as an argument to C or PL/SQL validation APIs or as a `Bind_ValidationRules` parameter as described in How to Set the `Bind_ValidationRules` Parameter.

Because the names of the segment columns that the customer will use for the code combinations are not known during development, you must use the lexical references listed in Table 26-1 to refer to the segment column and value attributes in the rule's `WHERE` clause. In addition to the lexical references, the `FLEXFIELD.VALIDATION_DATE` bind variable can be used in validation rule `WHERE` clauses. No other flexfield bind variables can be used.

### Table 26-1 Lexical References

<table>
<thead>
<tr>
<th>Lexical Type</th>
<th>Lexical Code</th>
<th>Example</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE</td>
<td>VALUE</td>
<td>&amp;{VALUE.VALUE}</td>
<td>Represents the VALUE column in segment lists of values and represents the segment column in combination lists of values.</td>
</tr>
<tr>
<td>VALUE_ATTRIBUTE</td>
<td>value attribute code</td>
<td>&amp;{VALUE_ATTRIBUTE.GL_ACCOUNT _TYPE}</td>
<td>Represents the value table value attribute column in segment lists of values and represents the combination table value attribute column in combination lists of values.</td>
</tr>
</tbody>
</table>

For example, if the following `WHERE` clause were registered as a validation rule for a segment, then the derived SQL query to retrieve the segment's list of values would be similar to Example 26-1, and the derived SQL query to retrieve the combination list of values would be similar to Example 26-2.
Tip:
To learn how to generate documentation about using the FND_FLEX_KF_SETUP_APIS PL/SQL package, see What You May Need to Know About the Key Flexfield Setup API.

Example 26-1  SQL Query to Retrieve Segment’s List of Values

```sql
SELECT ...
FROM fnd_vs_values_vl fvvv
WHERE fvvv.value_set_id = :Bind_ValueSetId
AND fvvv.value like :Bind_Value
AND (GL_AFF_AWC_API_PKG.gl_valid_flex_values(
    :Bind_ValidationDate, fvvv.value) = 'Y'))
```

Example 26-2  SQL Query to Retrieve Combination List of Values

```sql
SELECT ...
FROM gl_code_combinations glcc
WHERE glcc.chart_of_accounts_id = :Bind_SIN
AND ...
AND (GL_AFF_AWC_API_PKG.gl_valid_flex_values(
    :Bind_ValidationDate, glcc.segment5) = 'Y'))
```

26.2.1.4.2 How to Set the Bind_ValidationRules Parameter

Edit the view accessor's Bind_ValidationRules parameter to specify the validation rules to be applied to constrain the code-combination filters.

To set the Bind_ValidationRules parameter:

1. Create the view accessor and open it for editing as described in How to Define Code Combination Constraints.

2. In the Edit View Accessor dialog, set the Bind_ValidationRules value.

   The value of Bind_ValidationRules should be a semicolon-separated list of rule codes; for example:

   ```sql
   VALIDATION_RULE1;VALIDATION_RULE2
   ```

   The validation rules are predefined as part of the key flexfield definition. The supplied list is the list of rules must be applied when searching for a code combination.

   Note the following cautions when constructing this list:

   - The validation rule codes are case-sensitive.
   - Space characters are preserved. For example, "VRULE1; VRULE2" will be parsed into "VRULE1" and " VRULE2" (with a leading space).
   - Unrecognized and unused rules are discarded silently. For example, if you have a validation rule for an optional label, and the label has not been assigned yet in the current flexfield definition, then the rule will be ignored at runtime.
In a search user interface, the supplied validation rules also affect the list of values of a segment. For example, the end user may pick a value for a segment from a list of values, then use the segment value to search for a code combination. The list of values of the segment will be constrained by the supplied validation rules.

26.2.1.5 Enable or Disable Dynamic Combination Creation for a Specific Usage

You can use the `Bind_DynamicCombinationCreationAllowed` parameter to control the runtime entry of new code combinations for a key flexfield usage. This constraint type takes effect only when the key flexfield allows dynamic combination insertion. For more information, see Enabling Dynamic Combination Insertion.

To set the `Bind_DynamicCombinationCreationAllowed` parameter:

1. Create the view accessor and open it for editing as described in How to Define Code Combination Constraints.
2. In the Edit View Accessor dialog, set the `Bind_DynamicCombinationCreationAllowed` value.

   The value of `Bind_DynamicCombinationCreationAllowed` can be TRUE, FALSE, or null.

   By setting this value to TRUE or FALSE, you can control whether your specific usage of the key flexfield allows dynamic insertion even though the key flexfield as a whole is enabled for dynamic insertion. Set the value to TRUE if you want your usage of the key flexfield to allow dynamic insertion. Set the value to FALSE if you do not want your usage of the key flexfield to allow dynamic insertion. Set the value to null to indicate that the key flexfield itself should determine whether dynamic combination insertion is allowed or not.

   If the key flexfield does not allow dynamic combination insertion, this constraint is ignored. `Bind_DynamicCombinationCreationAllowed` is a row-level bind parameter.

26.2.2 How to Access Segment Labels Using the Java API

Segment labels (previously known as key flexfield qualifiers) that have been assigned to segments by customers can be accessed programmatically. The information can be accessed using the flexfield application module or the flexfield view row.

Example 26-3 is an example of Java code that retrieves segment label information from a deployed flexfield using the flexfield application module.

Example 26-4 is an example of Java code that retrieves segment label information from a deployed flexfield using the flexfield view row.

For more information about segment labels, see How to Implement Key Flexfield Segment Labels. For more information about the Java API, see the Javadoc.

Example 26-3 Retrieving Segment Label Information Using the Flexfield Application Module

```java
// First find the flexfield application module:
FlexfieldApplicationModuleImpl flexAM = (FlexfieldApplicationModuleImpl) rootAM.findApplicationModule("Kff1nAM1");

// Find the attributes labeled as "SEGMENT_LABEL_G1" in structure
// "VS_FRM Chr_ON_Chr".
// If you wish to use a structure instance number, you must further
```
// cast the application module into KFFApplicationModuleImpl and call
// getStructureCode(long) to find the code.
// For example,
// KFFApplicationModuleImpl kffAM = (KFFApplicationModuleImpl) flexAM;
// String code = kffAM.getStructureCode(12345);
List<AttributeDef> attrs = flexAM.getLabeledAttributes("VS_FRM_CHR_ON_CHR",
                "SEGMENT_LABEL_G1");
for (AttributeDef attr: attrs)
{
    System.out.println(attr.getName());
    // You can get the segment code through a static method.
    System.out.println(FlexfieldViewDefImpl.getSegmentCode(attr));

    // If you somehow need to construct a WHERE clause using this attribute,
    // this is the identifier you should use.
    System.out.println(attr.getColumnNameForQuery());

    // You can find the "column name" defined in the entity. The column name
    // is typically the database column name.
    System.out.println(attr.getColumnName());
}

Example 26-4    Retrieving Segment Label Information Using the View Row

// This is just for illustration. In a real application, the
// flexfield view row should be retrieved through the view link accessor.
ViewObject vo = rootAM.findViewObject("Kff1nAM1.DefaultFlexViewUsage");
vo.executeQuery();
while (vo.hasNext())
{
    FlexfieldViewRowImpl row = (FlexfieldViewRowImpl) vo.next();
    // Given a KFF view row, you can find the labeled attributes through
    // / the view def. An empty list is returned if the given label is not used
    // in the row.
    List<AttributeDef> labeledAttrs =
        row.getFlexfieldViewDef().getLabeledAttributes("SEGMENT_LABEL_RU1");
    for (AttributeDef attr: labeledAttrs)
    {
        System.out.print(attr.getName() + "=" +
            row.getAttribute(attr.getName()) + ";");
    }
    System.out.println();
}

26.2.3 How to Prepare Key Flexfield Business Components for Oracle Business
Intelligence

Oracle Business Intelligence is a comprehensive collection of enterprise business
intelligence functionality that provides the full range of business intelligence
capabilities including interactive dashboards, full ad hoc, proactive intelligence and
alerts, enterprise and financial reporting, real-time predictive intelligence, and more.

While key flexfields are modeled using polymorphic view objects, flexfield technology
is not compatible with Oracle Business Intelligence, which also requires reference data,
such as lookups, to be modeled as view-linked child view objects. For a key flexfield to
be used by Oracle Business Intelligence, it must be flattened into a usable static form.
To make this possible, you must designate the flexfield as business intelligence-enabled.
When you create business components for this key flexfield, the business
cOMPONENT modeler recognizes the business intelligence-enabled setting, and a view
object that is flattened for business intelligence is generated alongside the standard key flexfield polymorphic view object. You must also slightly modify the process of creating key flexfield view links and application modules.

When the business intelligence-enabled and flattened key flexfield is configured as part of an application, the implementor can select which individual flexfield segments to make available for use with Oracle Business Intelligence. Only the segments that are business-intelligence enabled are included in the flattened view object. Administrators must then import the changes into the Oracle Business Intelligence repository to make them available for Oracle Business Intelligence.

### 26.2.3.1 Enable a Key Flexfield for Oracle Business Intelligence

If you want customers to be able to do business intelligence queries on whatever segments they configure for a flexfield, you must business-intelligence enable the flexfield and its segments. A flattened view object is generated only if the key flexfield is business-intelligence enabled. A segment is included in the flattened view object only if the segment is business-intelligence enabled.

You can set the flexfield’s business intelligence-enabled flag at registration time using the `fnd_flex_kf_setup_apis.create_flexfield(…)` procedure, or you can set the flag later using the `fnd_flex_kf_setup_apis.update_flexfield(…)` procedure. To learn how to generate documentation about using these procedures, see What You May Need to Know About the Key Flexfield Setup API.

You can optionally provide flattened fact names for the flexfield’s entity details. This helps to automate the process for importing the key flexfield into Oracle Business Intelligence. To provide the flattened fact name, set the `BI_FLATTENED_FACT_NAME` value when you register the entity details using the `create_adfbc_usage(…)` procedure. You can also set the flag later using the `update_adfbc_usage(…)` procedure.

Use the Manage Key Flexfields task, which is accessed from the Setup and Maintenance work area of any Oracle Fusion Setup application, to enable the segments for business intelligence. Only the segments that are business-intelligence enabled are included in the flattened view object.

If you want to map the business intelligence-enabled segments to logical dimensions in the Oracle Business Intelligence logical model, use the Manage Key Flexfields task to create segment labels and to map the labels to the logical dimensions. Then assign the labels to the appropriate flexfield segments. By mapping the segments to the dimensions, you minimize the steps for importing the flexfield into Oracle Business Intelligence.

### 26.2.3.2 Produce a Flattened Model for a Business Intelligence-Enabled Key Flexfield

When you create business components for a business intelligence-enabled key flexfield, the business component modeler recognizes the business intelligence-enabled setting, and a view object that is flattened for Oracle Business Intelligence is generated alongside the standard key flexfield polymorphic view object. You must also slightly modify the process of creating key flexfield view links and application modules.

**Before you begin:**

1. Enable the flexfield and the desired segments for Oracle Business Intelligence as described in Enabling a Key Flexfield for.
2. If your flexfield will use hierarchical (tree structured) value sets, create column-flattened versions of the affected view objects and import them into your project before continuing.

If the flattened tree view objects are not in your project, the Create Flexfield Business Components wizard will report the missing view objects as errors.

For more information, see About Designing a Column-Flattened View Object for Oracle Business Intelligence.

To produce a flattened model for a business intelligence-enabled key flexfield:

1. Create key flexfield business components as described in How to Create Key Flexfield Business Components.

When a flexfield is business intelligence-enabled, the Create Flexfield Business Components wizard generates a business intelligence-specific view object and other business components under a directory called `analytics` in the package root directory. These are generated in addition to the typical key flexfield view object. The business intelligence-specific view object is distinguished from the typical key flexfield view object by the "BI:" prefix.

2. Create a view link using the procedure described in How to Create Key Flexfield View Links. Keep the following in mind:

   - The view object that you use for the source view object can be the same source view object that you used for the base key flexfield.
   - Create the view link from the source view object to the business intelligence-specific view object, which is the view object with the "BI:" prefix as shown in Figure 26-3.

   ![Figure 26-3 Create Flexfield View Link Wizard — View Objects Page](image)

3. Create an application module for use with Oracle Business Intelligence, as described in How to Create the Maintenance Application Module. Make the following changes:
a. On the Data Model page of the Create Application Module wizard, when you create an instance of the master view object, there is no need for a child view object.

b. On the Application Modules page of the wizard, add an instance of the key flexfield Oracle Business Intelligence application module as a nested instance of this application module. You can identify the Oracle Business Intelligence application module by the `analytics` subpackage under the package root.

**Note:**
If you already have a product Oracle Business Intelligence application module, you may use it.

4. Define the custom properties required to link the master view object instance to the default view instance.

Do this in the General navigation tab of the nested instance definition of the business intelligence-enabled flexfield application module, as shown in Figure 26-4.

**Figure 26-4  Custom Properties for Business Intelligence-Enabled Application Module**

As you do this, keep the following points in mind:

- The default view instance inside the business intelligence-enabled flexfield application module is typically called `DefaultFlexViewUsage`.
- The custom property names should be formatted as `BI_VIEW_LINK_mypropertyname`.
- The custom property values should be formatted as `source_viewobjectinstance_name, viewlink_definition_name, destination_viewobjectinstance_name`.
- Use the fully qualified view object instance names for the source view object and destination view object, and the fully qualified package name for the view link definition.
• Business intelligence joins between the view object instances you specify in different application modules are created during import from Oracle Application Development Framework (Oracle ADF).

### 26.2.4 How to Publish Key Flexfield Application Modules as Web Services

You can make access to a key flexfield available through web services, which will enable you to perform create, read, update, and delete (CRUD) operations on the flexfield data rows. You accomplish this by exposing a key flexfield application module as a web service and adding support utility methods for the flexfield service data object to the product application module.

When you generate a flexfield business component, the key flexfield business component and other artifacts are developed based on the information in the flexfield metadata. As illustrated in Figure 25-2, a base view object is created for the code combination id (CCID) and structure instance number (SIN) segments. If any contexts have been configured, subtype view objects are generated for each configured context.

As an example, suppose that an application module has the master view object Fkt1 and a view link from the master view object to the detailed key flexfield view object, Global1, which is a polymorphic view object.

The Business Component Browser view shown in Figure 26-5 corresponds to a particular row in the master view object, displaying the segment structure in the key flexfield with a SIN of 11.

**Figure 26-5  Business Component Browser View of Flexfield Row with SIN = 11**

The Business Component Browser view shown in Figure 26-6 corresponds to a different row in the master view object, displaying the segment structure in the key flexfield with a SIN of 25.
To make a key flexfield accessible through a web service:

1. Expose the key flexfield application module as a web service.
2. Test the web service.

26.2.4.1 Expose a Key Flexfield Application Module as a Web Service

You make key flexfield access available through web services by doing the following:

1. Setting a custom property for the flexfield view link.
2. Adding a transient attribute to the master view object to store the concatenated flexfield key.
3. Service-enabling the master view object.
4. Creating the service interface for the product application module within which the key flexfield application module is nested.
5. Adding flexfield service data object support utility methods to the product application module.

**Note:**

In this section, master view object refers to the view object for the reference table as illustrated by Figure 25-1.
Before you begin:

1. Create the master entity object and view object for the product table that references the key flexfield.

2. Create the flexfield business component as described in Building a Read-Only Reference Model.

   **Note:**
   When you generate a flexfield business component, JDeveloper automatically service-enables the business component by generating a Service Data Object (SDO) for the base view object and for every subtype view object.

3. Create a flexfield view link between the master view object and the flexfield business component as described in How to Create Key Flexfield View Links.

4. Nest the key flexfield application module instance in the product application module as described in How to Nest an Instance of the Key Flexfield Application Module in the Product Application Module.

5. Add the key flexfield view object instance to the application module as described in How to Add an Instance of a Key Flexfield View Object to the Product Application Module.

To expose a key flexfield application module as a web service:

1. Complete the following steps to ensure that Service Data Objects (SDOs) exist for all subtype objects.
   a. In the Application Navigator verify that .xsd files exist for all flexfield subtype view objects.
   b. Open the .xsd file for the key flexfield’s base view object and verify that <include> elements exist for all the flexfield subtype view objects.

   **Figure 26-7  The Include Elements for the Flexfield Subtype SDOs**

   c. If any subtype view object SDOs are missing, edit the key flexfield’s base view object, and, on the Java navigation tab, click the Edit Java options icon.
In the Select Java Options dialog shown in Figure 26-8, select Generate Service Data Object Class, ensure that the namespace is the same location that contains the flexfield view object XML files, and click OK.

When the SDO is generated for the base view object of the key flexfield polymorphic view object, generic SDOs are automatically generated for all the base view object subtypes.

**Figure 26-8 Generating the SDO for the Key Flexfield Base View Object**

2. Edit the view link between the master view object and the flexfield view object.

3. Click the General navigation tab in the overview editor, expand the Custom Properties section, and add a SERVICE_PROCESS_CHILDREN property set to true, if one does not already exist.

4. Edit the master view object.

5. In the overview editor, add a transient attribute to store the key flexfield concatenated string.
   a. Click the Attributes navigation tab, and click the Add icon in the Attributes section.
   b. In the View Attribute dialog shown in Figure 26-9, enter a name for the attribute.
c. Set the Java attribute type to **String**.

d. Leave the **Mapped to Column or SQL** checkbox unselected. A transient attribute does not include a SQL expression.

e. Select the **Always** radio button.

f. Leave the **Expression** blank.

g. Click **OK**.

6. Click the **Java** navigation tab and click the **Edit** icon in the **Java Classes** section to generate classes for the master view object.

7. In the Select Java Options dialog shown in **Figure 26-10**, select the following checkboxes and click **OK**:

   - Generate View Object Class
   - Include bind variable accessors
   - Include custom Java data source methods
   - Generate View Row Class
   - Include accessors
   - Generate View Object Definition Class
   - Generate Service Data Object Class
8. In the Java navigation tab, click the link for the View Row Class to open it in the editor.

9. Add the code shown in bold in Example 26-5 to the setter method for the transient attribute that you created. Set the viewAccessorName to the name of the view accessor from the view link between the master view object and the key flexfield view object.

The added code stores the key flexfield concatenated string.

10. Edit the product application module in which the key flexfield application module instance, master view object instance, and flexfield view object instance are nested.

11. Click the Web Service navigation tab and click the Add icon to enable support for the service interface. If the icon is not enabled, click the Edit icon instead and edit the pages that are named in the following steps.

12. In the Service Interface page of the Create Service Interface wizard, the Web Service Name and Target Namespace fields are automatically populated with appropriate values for this application module.

Click Next to continue.

13. In the Service Custom Methods page, select the client methods in the Available list that you want to expose as part of the service interface and move them to the Selected list.
14. Click **Next** to continue.

15. In the Service View Instances page select the view objects in the **Available** list that you want to expose as part of the service interface and move them to the **Selected** list.

16. Highlight each view object on the **Selected** list to display the **Basic Operations** and **View Criteria Find Operations** lists for the operations that are available for that view object, as shown in Figure 26-11.

Select the checkbox for each operation of the view object that you want to expose in the service interface and clear the rest.

**Figure 26-11  Create Service Interface Wizard — Service View Instances Page**

17. Click **Next** to continue.

18. In the Summary page, review your choices and click **Finish** to generate the web service from the application module. You should see that the **Web Service** navigation tab now reflects the custom methods, view instances, basic operations, and view criteria find operations that you chose to expose, as shown in Figure 26-12.
19. Click Finish.

The generated service interface components appear below the application module in the Application Navigator, as shown in Figure 26-13.

Figure 26-13  Application Module Service Interface Structure

20. In the overview editor for the product application module, click the Java navigation tab.

21. If the Application Module Class and the Application Module Definition Class do not appear in the list of Java classes, click the Edit icon and generate the classes.
22. In the Java Classes section, click the link for the Application Module Class.

23. Add the utility methods shown in Example 26-6 to the application module class. These utility methods enable web service clients to obtain FlexfieldSdoSupport objects to access the flexfield's information.

Replace Flexfield with the appropriate string for the flexfield that you are working with. Use a string that describes how the flexfield will be used by the customers. For example, getLedgerSdoNamespaceAndName is better than getGLSubtypeSdoNamespaceAndName.

In the getFlexfieldSdoSupport and getFlexfieldStructureInstanceNumber methods, replace getKffMAM1 with the name of the getter method for the nested maintenance application module instance.

24. In the overview editor for the product application module, click the Web Service navigation tab for the product application module and click the Edit icon in the Service Interface Custom Methods section.

25. In the Service Custom Methods page, move the newly added public methods to the Selected list to make them available for clients and click OK.

The application module's remote server implementation class will be modified to expose these methods.

**Example 26-5  Setter Method for the Transient Attribute**

```java
/**
 * Sets <code>value</code> as the attribute value for the calculated attribute KffConcatSegment
 * @param value value to set the KffConcatSegment
 */
public void setKffConcatSegment(String value) {
    String concatAttrPrefix = FlexfieldProperty.PREFIX;
    String concatAttrPostfix = FlexfieldProperty.CONCATENATED_STORAGE_ATTR_POSTFIX;
    // Modify next line to set viewAccessorName to name of the VL
    String viewAccessorName = "put KFF view link accessor name here";
    String concatAttrName = concatAttrPrefix + viewAccessorName + concatAttrPostfix;
    setAttributeInternal(concatAttrName, value);
    setAttributeInternal(KFFCONCATSEGMENT, value);
}
```

**Example 26-6  Utility Methods for Flexfield Service Data Object Support**

```java
// Change method name as appropriate
public List<String> getFlexfieldSdoNamespaceAndName(
    String structureInstanceCode) {
    FlexfieldSdoSupport ss = getFlexfieldSdoSupport(structureInstanceCode);
    if (ss == null) {
        return null;
    }
    return Arrays.asList(ss.getSdoNamespace(), ss.getSdoName());
}

// Change method name as appropriate
public String getFlexfieldSdoPath() {
    FlexfieldSdoSupport ss = getFlexfieldSdoSupport(null);
    if (ss == null) {
        return null;
    }
    return null;
}
```
```java
return null;
}
return ss.getDiscriminatorSdoPath();
}

// Change method name as appropriate
public List<String> getFlexfieldSegmentSdoPaths(String structureInstanceCode,
                                             List<String> segmentCodes) {
    FlexfieldSdoSupport ss = getFlexfieldSdoSupport(structureInstanceCode);
    if (ss == null) {
        return null;
    }
    ArrayList r = new ArrayList(segmentCodes.size());
    for (String segmentCode : segmentCodes) {
        r.add(ss.getSegmentSdoPath(segmentCode));
    }
    return r;
}

// Change method name as appropriate
private FlexfieldSdoSupport getFlexfieldSdoSupport(String structureInstanceCode) {
    /**
     * @param structureInstanceCode set to null to get the parent (base)
     */
    // Find the nested maintenance application module instance
    KFFMApplicationModuleImpl am;
    // Change getKffMAM1 to name of the getter method for the nested KFF AM
    am = (KFFMApplicationModuleImpl) getKffMAM1();
    return am.getSdoSupport(structureInstanceCode);
}

// Change method name as appropriate
public Long getFlexfieldStructureInstanceNumber(String structureInstanceCode) {
    // Find the maintenance application module instance
    KFFMApplicationModuleImpl am;
    // Change getKffMAM1 to name of the getter method for the nested KFF AM
    am = (KFFMApplicationModuleImpl) getKffMAM1();
    return am.getStructureInstanceNumber(structureInstanceCode);
}
```

### 26.2.4.2 Test the Web Service

You can test the key flexfield web service access by providing web server connection information, deploying and manually testing the web service, and optionally writing Java client programs to call the support utility methods for the flexfield service data object to test the service.

**Before you begin:**

1. Ensure that the BC4J Service Client and BC4J Service Runtime libraries are included in your project.
2. Create a writable maintenance model as described in [Building a Writable Maintenance Model](#).
3. Expose the key flexfield maintenance application module as a web service as described in Exposing a Key Flexfield Application Module as a Web Service.

To test the web service:

1. Expand Application Resources > Descriptors > ADF Meta-INF, and open the connections.xml file.

2. Locate the Reference element for the product application module's service (ApplicationService in this example).

   This is the service that you created in Exposing a Key Flexfield Application Module as a Web Service for the product application module in which the key flexfield maintenance application module instance, master view object instance, and flexfield view object instance are nested.

3. Add the StringRefAddr elements that are shown in bold in Example 26-7. Modify the host and port number in the jndiProviderURL entry to point to an instance of Oracle WebLogic Server. The port number is typically 7101.

4. Run the remote server class for the product application module to deploy the service to an Integrated WebLogic Server instance and to manually test the web service.

   **Note:**
   The remote server class was generated when you exposed the key flexfield as a web service in Exposing a Key Flexfield Application Module as a Web Service. This class has a name that ends with ServiceImpl.java.

5. Optionally, create and run Java client programs to test invoking the web service.

   Example 26-8 is an example of how a client test program would use the support utility methods for the flexfield service data object that you added in Exposing a Key Flexfield Application Module as a Web Service.

**Example 26-7 StringRefAddr Elements to Add to Application Module Reference in connections.xml**

```xml
<Reference name="{http://xmlns.oracle.com/oracle/apps/fnd/appcore/flex/test/kff1/model/}ApplicationService"
   className="oracle.jbo.client.svc.Service" xmlns="">
   <Factory className="oracle.jbo.client.svc.ServiceFactory"/>
   <RefAddresses>
       <StringRefAddr addrType="serviceInterfaceName">
           <Contents>oracle.apps.fnd.appcore.flex.test.kff1.model.ApplicationService</Contents>
       </StringRefAddr>
       <StringRefAddr addrType="serviceEndpointProvider">
           <Contents>ADFBC</Contents>
       </StringRefAddr>
       <StringRefAddr addrType="jndiName">
           <Contents>ApplicationServiceBean#oracle.apps.fnd.appcore.flex.test.kff1.model.ApplicationService</Contents>
       </StringRefAddr>
   </RefAddresses>
</Reference>
```

---

26-24 Oracle Fusion Applications Developer's Guide
Example 26-8  Sample Java Code to Test the Web Service

```java
public void testSDOAPis()
{
    // Flexfield service data objects are created automatically based on the
    // flexfield definition. Certain information such as segment codes must be
    // transformed to be used in a service data object definition.
    AcctKffMaintService acctKffMaintService =
        (AcctKffMaintService)
            ServiceFactory.getServiceProxy(AcctKffMaintService.NAME);
    DataFactory dataFactory =
        ServiceFactory.getDataFactory(acctKffMaintService);

    // Get the namespace and name corresponding to a particular
    // structure instance code.
    List<String> accountSdoInfo =
        acctKffMaintService.getAcctSdoNamespaceAndName("CC_ACCT_LOC_PRJ_SI");
    System.out.println(accountSdoInfo);
    DataObject accountDo =
        dataFactory.create(accountSdoInfo.get(0), accountSdoInfo.get(1));
    System.out.println(accountDo);
    accountDo.set acctKffMaintService.getAcctSDOPath(),
        acctKffMaintService.getAcctStructureInstanceNumber("CC_ACCT_LOC_PRJ_SI");

    // Get segment paths for attributes COST_CENTER and LOCATION.
    List<String> segmentPaths =
        acctKffMaintService.getAcctSegmentSdoPaths("CC_ACCT_LOC_PRJ_SI",
            Arrays.asList("COST_CENTER", "LOCATION");
    System.out.println(segmentPaths);

    //Update COST_CENTER and LOCATION after obtaining their segmentPaths.
    accountDo.set (segmentPaths.get(0), "A12");
    accountDo.set (segmentPaths.get(1), "UK");
}
```
26.2.5 How to Access Key Flexfields from an ADF Desktop Integration Excel Workbook

ADF Desktop Integration makes it possible to combine desktop productivity applications with Oracle Fusion applications, so you can use a program such as Microsoft Excel as an interface to access application data.

Using ADF Desktop Integration, you can incorporate key flexfields into an integrated Excel workbook, so you can work with the flexfield data from within the workbook.

For more general information about integrating Oracle Fusion applications with desktop applications, see the Oracle Fusion Middleware Developing Applications with Oracle ADF Desktop Integration guide. This guide provides most of the information you need to complete the required activities, including the following:

- Preparing your development environment for desktop integration.
- Creating a page definition file that will expose the Oracle ADF bindings for use in Excel.
- Creating an Excel workbook to integrate with the key flexfield.
- Preparing your Excel workbook to access the column containing the flexfield.
- Incorporating a key flexfield as a dynamic or static column in an ADF Desktop Integration Table on a worksheet in the workbook.

**Note:**

The Oracle Fusion Middleware Developing Applications with Oracle ADF Desktop Integration guide does not make explicit reference to technologies documented in this Developer's Guide, and this guide does not repeat the content in the Developing Applications with Oracle ADF Desktop Integration guide. Therefore, you must read the Developing Applications with Oracle ADF Desktop Integration guide for a full understanding of how to use ADF Desktop Integration technology in general.

In addition to the standard implementation steps covered in that guide, you must modify your implementation to accommodate flexfields, as discussed in the following sections.

There are two ways to access a key flexfield in Excel:

- Using a dynamic column in an ADF Desktop Integration Table.
  
  A web page in a popup dialog can be associated with a dynamic column, enabling end users to pick flexfield segment values. Alternatively, users can enter values directly into the segment fields. No custom code is required in either case.

  This is the most typical scenario. Each key flexfield segment is displayed as a distinct column in the ADF Desktop Integration Table. First you configure ADF Desktop Integration with a dynamic column key flexfield, and then, if necessary, you handle user-initiated structure code changes.

- Using a static column in a popup dialog associated with a single cell. Use this approach for either of the following reasons:
  
  - The key flexfield is in a non-table area of the worksheet.
The ADF Desktop Integration Table needs to expose the same key flexfield instance more than once. In this case, only one instance can be dynamic. All other instances should be exposed as static columns.

In addition to using the popup dialog, end users can enter values directly into the segment field, with the values separated by an appropriate delimiter that you specify.

**Note:**

A static column is any column for which the `DynamicColumn` property is set to `False`.

Individual flexfield segments do not appear in the worksheet. Instead, ADF Desktop Integration invokes a separate JSPX page on which the flexfield will be visible. You can use this scenario with an ADF Desktop Integration form, or either table type, by configuring ADF Desktop Integration with a static column key flexfield.

**Note:**

The titles of the popup dialog components must be set to the name of the key flexfield, such as "Account," to be consistent across Oracle Fusion Applications.

The key flexfield’s segments are part of your database table, so the flexfield is generated against the same entity object on which your worksheet view object is based.

In addition to configuring ADF Desktop Integration with the dynamic or static column key flexfield, you might also need to call a custom application module to handle the update or insert of a key flexfield data row.

### 26.2.5.1 Configure ADF Desktop Integration with a Dynamic Column Key Flexfield

When you configure the ADF Desktop Integration Table, make the following changes:

- Add the ADF Desktop Integration Model API library to your data model project.

- In your page definition for the worksheet, add the key flexfield that you want to access to the master worksheet view object as a child node. Do not add any display attribute to the child node, which expands as a dynamic column in the worksheet.

For more information about how to create a page definition file for a desktop integration project, see the *Working with Page Definition Files for an Integrated Excel Workbook* section of the *Developing Applications with Oracle ADF Desktop Integration*.

- To make the key flexfield column of the ADF Desktop Integration Table component dynamic, set the `DynamicColumn` property in the `TableColumn` array of the ADF Desktop Integration Table to `True`. A dynamic column in the `TableColumn` array is a column that is bound to a tree binding or tree node binding whose attribute names are not known at design time. A dynamic column can expand to more than a single worksheet column at runtime.
For more information about the binding syntax for dynamic columns, see the *Adding a Dynamic Column to Your ADF Table Component* section of the *Developing Applications with Oracle ADF Desktop Integration*.

- For the table’s **UpdateComponent** and **InsertComponent** properties, specify one of the following as the subcomponent to use:
  - Inputtext
  - OutputText
  - ModelDrivenColumnComponent
- For the subcomponent’s **Value** property, access the Expression Builder, expand the Bindings node and your tree binding for the table, and select the flexfield node.
- For the subcomponent’s **Label** property, access the Expression Builder, expand the Bindings node and your tree binding for the table, and select the flexfield node.

For information about the Expression Builder, see the *Using the Expression Builder* section in the *Developing Applications with Oracle ADF Desktop Integration*.

### 26.2.5.2 Handling User-Initiated Structure Code Value Changes in a Dynamic Column Key Flexfield

ADF Desktop Integration requires that to use a dynamic column implementation, the structure of the key flexfield should remain constant for all rows in a given result set. However, each time a new result set is downloaded into the table, the structure code value (and thus the structure) can be changed.

If the structure code value is set globally for the end user of the workbook, changes are not an issue. However, if the end user can control the structure code value (for example, using an LOV in a “header” form), your application must be able to respond appropriately to update the key flexfield structure.

After the end user specifies a structure code value, you must invoke the worksheet **UpSync** method to get the new value into the model. Then you can use the ADF Desktop Integration Table **Download** method to get fresh data with the new key flexfield structure.

**Note:**

For an insert-only table, the **Download** method is undesirable. For these cases, use either the ADF Desktop Integration Table **DownloadForInsert** method or the **Initialize** method to enable the ADF Desktop Integration Table component to reconfigure to accommodate the new flexfield structure.

### 26.2.5.3 Configure ADF Desktop Integration with a Static Column Key Flexfield

ADF Desktop Integration supports key flexfields by using tree bindings in an ADF Desktop Integration Table. If you are adding your key flexfield as a static column, you can alternatively use an ADF Read-Only Table. Keep in mind that ADF Read-Only Tables support static columns, but not dynamic columns. Popup dialogs support both types.
Use Key Flexfield Advanced Features in Reference Mode

Note:
A key flexfield appears as a node in the tree binding at design time. Because flexfields are built up dynamically at runtime, you will not see any attributes under the flexfield node in the page definition, but the node itself is present.

When you configure the popup dialog, make the following changes:

- You can use the column’s action set properties to make the key flexfield web page available for editing. You should include the attributes used in the web page in the table’s cached attributes unless the row will be committed immediately.

- You must choose a fixed attribute (the key flexfield CCID) to represent the flexfield in the worksheet. Add a Dialog action to the `DoubleClickActionSet` of an `InputText` or `OutputText` component, then connect the Dialog action to a JSPX page that will display the key flexfield.

For more information about how to create a page definition file for a desktop integration project, see the Working with Page Definition Files for an Integrated Excel Workbook section of the Developing Applications with Oracle ADF Desktop Integration.

For static display of a key flexfield in an ADF Desktop Integration workbook, you must create an updatable transient attribute in the view object on which the ADF Desktop Integration table is based. This transient attribute will hold the concatenated value of the key flexfield segments, separated by a delimiter. If one purpose of the worksheet is to display existing data from the database, the transient attribute should be populated using custom application module methods upon returning from a popup dialog or opening the worksheet.

26.2.5.4 Handle Update or Insert of a Key Flexfield Data Row

To handle updating or inserting a data row containing a key flexfield in an ADF Desktop Integration table, you call a custom application module method that contains appropriate code, as follows:

- To update an existing row, add your code to the `UpdateRowActionId` property of the table.

- To insert a new row, add your code to the `InsertAfterRowActionId` property of the table.

The following examples demonstrate the code needed to accomplish these tasks. Example 26-9 and Example 26-10 apply to an ADF Desktop Integration implementation with the key flexfield exposed as a dynamic column. Example 26-11 and Example 26-12 apply to an ADF Desktop Integration implementation with the key flexfield exposed as a static column.

**Example 26-9   Updating an Existing Row with a Key Flexfield Dynamic Column**

You add this code as an application module method that will be invoked from the `UpdateRowActionId` property of an ADF Desktop Integration Table. This code will be invoked for every row that is updated.

```java
Row tempRow = null;
// Get KFF child row information based on the KFF view link accessor.
KFFViewRowImpl acctRow;
ViewRowImpl kffAcctRow = (KFFViewRowImpl)linerow.getAccountLineKff();

// If it is not child row (for a new row case or cases where
```
// KFF data is not invalid/present for existing DB Row
// get a dummy row from ADF Desktop Integration helper class.

if (kffAcctRow == null) {
    tempRow = ModelHelper.getAdfdiTempChildRow(linerow, "AccountLineKff");
    kffAcctRow = (ViewRowImpl)tempRow;
}

Long kffAcctId = null;
String acctSeg=null;

// Check whether the KFF row is an instance of KFFViewRowImpl,
// which means you are updating valid KFF data.
// If not, it means you are creating new KFF data.
// Based on that, logic of deriving updated segments from worksheet differs.

if (kffAcctRow instanceof KFFViewRowImpl){
    acctRow = (KFFViewRowImpl)fkRow.getAccountLineKff();
    acctSeg = acctRow.getBufferedConcatenatedSegments();
} else {
    acctSeg = KFFViewRowImpl.getConcatenatedSegments(kffAcctRow);
}

kffAcctId = linerow.getKeyFlexfieldCombinationID("AccountLineKff",acctSeg);

// If you need dynamic insert (the creation of new segment combinations)
// from the ADF Desktop Integration worksheet, make sure the end user
// supplied a valid value for at least one segment.

if (kffAcctId==null) {
    // If there was not valid CCID obtained earlier, that means you
    // are trying to add a new combination.

    String delimiter = FlexfieldViewDefImpl.getDelimiter(kffAcctRow.getViewDef());
    List segments =
        FlexfieldViewDefImpl.getFlexfieldAttributes(kffAcctRow.getViewDef());
    StringBuffer delim = new StringBuffer();
    for (int i = 0; i < segments.size() - 1; i++) {
        delim.append(delimiter);
    }

    // If getConcatenatedSegments() returns only delimiter information,
    // that means end user has not supplied valid KFF Segment values.

    if (!acctSeg .equals(delim.toString())) {
        linerow.setKeyFlexfieldCombinationID("AccountLineKff",
            acctSeg);
    }
//Get CCID value based on segment information

kffAcctId =
    linerow.getKeyFlexfieldCombinationID("AccountLineKff", acctSeg );
}
//if kffAcctId is null

//setting CCID column with CCID value

    linerow.setDistCodeCombinationId(kffAcctId);
**Example 26-10  Inserting a New Row with a Key Flexfield Dynamic Column**

Add this code as an application module method that will be invoked from the InsertAfterRowActionId property of an ADF Desktop Integration Table. This code will be invoked for every row that is inserted.

```java
Row tempRow = null;
// Retrieve key flexfield child row information based on
// the KFF view link accessor
ViewRowImpl kffAcctRow = (ViewRowImpl)linerow.getAccountLineKff();

// If not a child row (for new row case or cases where
// KFF data is not invalid/present for existing DB Row),
// get a dummy row from ADF Desktop Integration helper class
if (kffAcctRow == null) {
    tempRow = ModelHelper.getAdfdiTempChildRow(linerow, "AccountLineKff");
    kffAcctRow = (ViewRowImpl)tempRow;
}
Long kffAcctId = null;
kffAcctId =
    linerow.getKeyFlexfieldCombinationID("AccountLineKff",
    KFFViewRowImpl.getConcatenatedSegments(kffAcctRow));

// If you need dynamic insert (creating new segment combinations)
// in the ADF Desktop Integration worksheet,
// make sure the end user supplies a valid value for at least one segment.
if (kffAcctId==null) {
    String delimiter = FlexfieldViewDefImpl.getDelimiter(kffAcctRow.getViewDef());
    List segments =
        FlexfieldViewDefImpl.getFlexfieldAttributes(kffAcctRow.getViewDef());
    StringBuffer delim = new StringBuffer();
    for (int i = 0; i < segments.size() - 1; i++) {
        delim.append(delimiter);
    }
    // If getConcatenatedSegments() return only delimiter information,
    // that means end user has not supplied valid KFF Segment values.
    if
        !KFFViewRowImpl.getConcatenatedSegments(kffAcctRow).equals(delim.toString())){
        linerow.setKeyFlexfieldCombinationID("AccountLineKff",
            KFFViewRowImpl.getConcatenatedSegments(kffAcctRow));
        // Get CCID value based on segment information.
        kffAcctId =
            linerow.getKeyFlexfieldCombinationID("AccountLineKff",
                KFFViewRowImpl.getConcatenatedSegments(kffAcctRow));
    }
}
// Set CCID column with CCID value.
linerow.setDistCodeCombinationId(kffAcctId);
```

**Example 26-11  Updating or Inserting a Row with a Key Flexfield Static Column**

This code should be added to the setter method of the transient attribute in your view object RowImpl.

```java
Example 26-11  Updating or Inserting a Row with a Key Flexfield Static Column
This code should be added to the setter method of the transient attribute in your view object RowImpl.
```
setAttributeInternal(TRANSIENTACCOUNT, value);

// Get KFF child row information based on the KFF view link accessor.

ViewRowImpl kffAcctRow = (ViewRowImpl)linerow.getAccountKff();
Row tempRow;

// If not child row (for new row case or cases where
// KFF data is not invalid/present for existing DB Row),
// Get a dummy row from ADF Desktop Integration helper class
if (kffAcctRow == null) {
    tempRow = ModelHelper.getAdfdiTempChildRow(this, "AccountKff");
    kffAcctRow = (ViewRowImpl)tempRow;
}
Long kffAcctId = null;

// If you need dynamic insert (creating new segment combinations)
// in the ADF Desktop Integration worksheet,
// make sure the end user supplies a valid value for at least one segment.

String delimiter = FlexfieldViewDefImpl.getDelimiter(kffAcctRow.getViewDef());
List segments =
FlexfieldViewDefImpl.getFlexfieldAttributes(kffAcctRow.getViewDef());
StringBuffer delim = new StringBuffer();
for (int i = 0; i < segments.size() - 1; i++) {
    delim.append(delimiter);
}

// If getConcatenatedSegments() returns only delimiter information,
// that means the end user has not supplied a valid segment value.

if (value!=null)
    if (!
        KFFViewRowImpl.getConcatenatedSegments(kffAcctRow).equals(delim.toString()))
        this.setKeyFlexfieldCombinationID("AccountKff",value);

    kffAcctId = this.getKeyFlexfieldCombinationID("AccountKff", value);
    //set your orignal attribute with CCID value.
    this.setAcctsPayCodeCombinationId(kffAcctId);
}


Example 26-12   Applying Modified Segment Values to a Cell in a Key Flexfield Static Column

You add this code as a custom application module method that will be invoked from
the ActionListener property of the OK button in the popup dialog JSPX page.

// Get a reference to your ADF DesktopIntegration table view object.
DesktopQuickInvoicesHeaderVOImpl headerVO =
    this.getDesktopQuickInvoicesHeader();

// Get a reference to the current row, which is the row from which
// popup dialog is opened
DesktopQuickInvoicesHeaderVORowImpl headerRow =
    (DesktopQuickInvoicesHeaderVORowImpl)headerVO.getCurrentRow();

// Get a reference to your key flexfield row
ViewRowImpl kffAcctRow = (ViewRowImpl)headerRow.getAccountKff();
Row tempRow;

// If that is null (for a null CCID or an invalid CCID or a new row),
26.3 Complete the Development Tasks for Key Flexfields in Secondary Mode

The most common use of a key flexfield is for a product table to have a foreign key to the primary key of the combinations table. This provides the flexibility of storing all the combinations in a single table and having references to these combinations from the product tables. However, there are circumstances where application developers might need to capture segment values in a transaction table or a setup table. In this case, the key flexfield becomes a data capturing tool, and the captured data is stored in a product table. There is no direct relationship between the product table and the key flexfields combinations table. This type of usage is called secondary usage.

There are two types of secondary usage:

- Single-segment secondary usage: Use this mode when you want to capture only one segment value in a transaction or setup table. For example, if you want to capture the default cost center value for an employee in the EMPLOYEES table in the DEFAULT_COSTCENTER column.

- All-segment secondary usage: Use this mode when you want to capture all the flexfield's segment values in a transaction or setup table. For example, if you want to capture all the segment values in a general ledger setup table for use in filling in missing values in the subledger account engine.

Note:

To incorporate a key flexfield secondary usage into your application, you must have already defined and registered the key flexfield primary usage on which it is based. See Registering and Defining Key Flexfields Using the Setup APIs, before continuing.

The development tasks for key flexfields in secondary mode consist of the following steps:

1. Complete the registration of a key flexfield secondary usage (all-segments or single-segment).

2. Create key flexfield business components that are based on the secondary usage for use in secondary mode development tasks.

3. Create a view link between your product view object and the secondary mode key flexfield.

4. The remainder of the development process is essentially the same as the consumer development process for key flexfield primary usages. You can skip the section on
creating key flexfield view links and continue with the tasks described in the following sections:

a. How to Nest an Instance of the Key Flexfield Application Module in the Product Application Module

b. How to Add an Instance of a Key Flexfield View Object to the Product Application Module

c. Employing Key Flexfield UI Components on a Page

Note:
This section contains additional information specific to key flexfield secondary usages.

d. How to Configure Key Flexfield UI Components

Note:
This section contains additional information specific to key flexfield secondary usages.

After you have completed the development tasks for secondary usages, you can incorporate the secondary usages in the application user interface as described in Employing Key Flexfield UI Components on a Page.

26.3.1 How to Register a Key Flexfield All-Segment Secondary Usage

All-segment secondary usages have a column in the product table for every segment column in the combinations table.

To register an all-segment secondary usage:

1. Add columns to your product table to represent all of the key flexfield segment columns in the combinations table. The columns that you add must match exactly in number, data type, and size, the corresponding columns in the combinations table.

   Furthermore, the column names must also be the same as in the combinations table, except for an optional prefix.

   For example, if the column names are A1 and A2 in the combinations table, then in the secondary usage they could again be A1 and A2, respectively, or with a prefix they could be X_A1 and X_A2. They cannot be B1 and Y_B2, nor any variation that does not end in the names of the combinations table columns.

2. Use the PL/SQL registration APIs in the FND_FLEX_KF_SETUP_APIS package to register the secondary usage.

   To learn how to generate documentation about using the APIs, see What You May Need to Know About the Key Flexfield Setup API.

3. Create an ADF Business Components usage for the flexfield table usage as described in Registering Entity Details Using the Setup APIs.
To implement a key flexfield secondary usage, you select the usage at design time. For more information, see *How to Create Key Flexfield Business Components*.

If you need to change a table usage after creating it, you must delete the table usage, then re-create it.

### 26.3.2 How to Register a Key Flexfield Single-Segment Secondary Usage

Single-segment secondary usages have one column in the product table to capture a single segment column in the combinations table.

**To register a single-segment secondary usage:**

1. Add the key flexfield segment column that you want to capture to your product table. The table cannot be the combinations table for the flexfield.

2. Define a segment label for the segment that you want to capture.

   The segment label must be defined as *Unique* to ensure that only one segment in a given structure can be associated with this label.

   For more information about segment labels, see *How to Implement Key Flexfield Segment Labels*.

3. Use the PL/SQL registration APIs in the `FND_FLEX_KF_SETUP_API` package to register the secondary usage.

   You must supply the `SEGMENT_LABEL_CODE` value to identify the unique segment label, and the `COLUMN_NAME` value to identify the column in your table in which the segment value will be stored.

   To learn how to generate documentation about using the APIs, see *What You May Need to Know About the Key Flexfield Setup API*.

4. Create an ADF Business Components usage for the flexfield table usage as described in *Registering Entity Details Using the Setup APIs*.

To implement a key flexfield secondary usage, you select the usage at design time. For more information, see *How to Create Key Flexfield Business Components*.

If you need to change a table usage after creating it, you must delete the table usage, then re-create it.

### 26.3.3 How to Create Key Flexfield Business Components for Secondary Usage

Zero or more secondary usages can be defined for a given flexfield, each one potentially on a different product table.

**Before you begin:**

1. One or more required libraries might have not been automatically included in your project. Ensure that all required libraries, notably the BC4J Service Runtime, Java EE 1.5 and Java EE 1.5 API libraries, are included.

2. Using the Create Entity Object wizard, create entity objects for the combinations tables that you have defined. Verify the following:

   - At least one customization class is included in the `adf-config.xml` file.
Note:
This serves to ensure correct application behavior. It does not matter which customization class you include.

For information about customization layers, see the "Understanding Customization Layers" section in the Extensibility Guide for Developers.

- These entity objects are directly modeled on the combinations tables; hence, they contain the fixed (nonflexfield) columns, if any, along with all of the flexfield columns. In general, all columns should be included.
- The entity objects have primary keys defined.
- The CCID column is of type data type `java.lang.Long`.
- The SIN column, if it exists, is of data type `java.lang.Long`.

Caution:
The SIN attribute cannot be transient with a calculated value. It can be based on a database table column, or it can be SQL-derived.

- The `NUMBER` type segment columns are of data type `java.math.BigDecimal`.
- The `VARCHAR2` type segment columns are of data type `java.lang.String`.
- The package name and the object name prefix for each entity object are registered with the flexfield usage to which it will provide data, as described in Registering Entity Details Using the Setup APIs.

3. Build your project to ensure that the entity objects are available in classes. The modeler relies on what is in your classes.

To create key flexfield business components for a secondary usage:

1. In the Application Navigator, right-click the project and choose New.
2. In the New Gallery, expand Business Tier, select ADF Business Components and then Flexfield Business Components, and click OK.
3. On the Flexfield page of the Create Flexfield Business Components wizard, select key from the Type dropdown list as shown in Figure 26-14.
4. In the **Application** field, specify the full name of the application to which your key flexfield belongs.

   You can browse for the name, and filter by **ID**, **Short Name**, or **Name**.

5. In the **Code** field, specify the code of the key flexfield you want to use.

   You can browse for and filter by **Code**.

6. In the **Usage** section, select the table row that contains the desired secondary key flexfield usage. Key flexfield usage can be one of the following types:

   - An all-segment secondary usage of the key flexfield on a product table other than the combinations table. Zero or more all-segment secondary usages can be defined for a given flexfield, each one potentially on a different product table. You can identify all-segment secondary usages by the presence of the prefix (Partial) in the **Description** field.

   - A single-segment secondary usage of the key flexfield on a product table other than the combinations table. Zero or more single-segment secondary usages can be defined for a given flexfield, each one potentially on a different product table. You can identify single-segment secondary usages by the presence of the prefix (Partial Single) in the **Description** field.

   Do not select a flexfield usage without a prefix in the **Description** field. For more information about key flexfield secondary usages, see **How to Register a Key Flexfield All-Segment Secondary Usage**.

7. Click **Next**.

8. On the Path page, select the path of the directory structure in which to create the flexfield business components and click **Next**.

9. On the Entity Object page, select the entity object to use as the data source for the key flexfield **Figure 26-15**.
Because you selected a secondary usage on the Flexfield page, you must select the entity object for the table where that usage is defined.

The entity object you select must include all of the attributes that will be referenced by the flexfield. For secondary usages, this includes attributes that represent the SIN column, the DSN column if it exists in the combinations table, and all of the flexfield segment columns.

**Note:**
If you select a polymorphic entity object, ensure that the InheritPersonalization property for every subtype entity is set to true.

**Caution:**
The Create Flexfield Business Components wizard is case-sensitive. All column names — and the names of the flexfield entity object attributes associated with them — must be uppercase.

---

**Figure 26-15** Create Flexfield Business Components Wizard — Entity Object Page

10. You might wish to select an entity object for which the key flexfield attributes are defined as transient (not based on database table columns). If you need to do this, then select The names of the flexfield attributes match the names of registered columns exactly; do not check the underlying columns.

When a key flexfield entity object attribute is transient, there is no matching underlying column name. When you select this option, the system will match the entity object attribute names to the key flexfield column names, and use the matching attributes to access the flexfield data. Ensure that the entity object has a full set of attributes with matching names before you select this option.
**Caution:**

The transient SIN attribute cannot be a calculated value; it must be SQL-derived (computed using a SQL expression).

This entity object must be registered under the primary usage as described in Registering Entity Details Using the Setup APIs. There is no need to register another table for this purpose, even if the entity object is based on some other table.

**Note:**

If the entity object with transient key flexfield attributes is not based on the primary usage, the transient attributes must be named using the same prefix as the other attributes of that entity object (and the corresponding table columns). For more information, see How to Register a Key Flexfield All-Segment Secondary Usage.

11. Click Next.

12. On the Usage Settings page, select **Mark All Segments Optional** as shown in Figure 26-16 if this secondary usage is simply to add extra information for a product row and the user should not be required to provide information for every segment.

![Figure 26-16 Create Flexfield Business Components Wizard — Usage Settings Page](image)

13. From the **Structure Instance Number** dropdown list, select the entity attribute that corresponds to the key flexfield SIN for the secondary usage. The SIN must be an attribute of type `java.lang.Long`.

If the key flexfield is data set-enabled, this page will also contain a **Data Set Number** dropdown list. From the dropdown list, select the entity attribute that
corresponds to the DSN for the secondary usage. The DSN must be an attribute of type `java.lang.Long`.

**Note:**
Each secondary usage must have its own SIN attribute. If multiple secondary usages must share the same structure instance number, you must create multiple SQL-derived or transient attributes that provide the same structure instance number. Use the entity object Java class to synchronize all SIN attributes if the structure instance number for these usages is not read-only.

**14. Click Next.**

**15. The Naming page displays the entity object's package name and object name that you registered for the usage, as described in Registering Entity Details Using the Setup APIs. Click Next.**

**16. On the Summary page, review your choices and click Finish.**

The business components generated will replace any existing ones that are based on the same flexfield.

**Note:**
This wizard might fail with a "ClassNotFound" exception message. This indicates that one or more required libraries have not been automatically included in your project, notably the BC4J Service Runtime, Java EE 1.5 and Java EE 1.5 API libraries. You can resolve this issue by manually adding any missing libraries; then you can complete this procedure successfully.

**17. Refresh the project to see the newly created flexfield business components in the Application Navigator.**

### 26.3.4 How to Create Key Flexfield View Links for a Secondary Usage

A view link is needed whenever a product view object references your key flexfield. The base view object can have many incoming view links from various product view objects, as a key flexfield is usually shared by many product tables.

**Before you begin:**
You should have already created a master view object for your entity object using the standard wizard.

Ensure that the view object does not include flexfield attributes such as `SEGMENT1VARCHAR2, SEGMENT2NUMBER`, and so on. Ensure that you include the attributes that are needed for the foreign key reference, such as `CCID, SIN`, and, if present, `DSN`. Ensure that the CCID attribute's Display control hint is set to Hide.

**To create a key flexfield view link for a secondary usage:**

1. In the Application Navigator, right-click the project and choose **New**.
2. In the New Gallery, expand **Business Tier**, select **ADF Business Components** and then **Flexfield View Link**, and click **OK**.

3. In the Create Flexfield View Link wizard, on the Name page, provide a package name as shown in **Figure 26-17**.

![Figure 26-17 Create Flexfield View Link Wizard — Name Page](image)

4. Enter a name for the secondary mode view link.

5. Click **Next**. The View Objects page appears, as shown in **Figure 26-18**.

![Figure 26-18 Create Flexfield View Link Wizard — View Objects Page](image)

6. In the **Select Source View Object** tree, select a secondary usage view object.

7. In the **Select Destination Flexfield** tree, expand the available flexfield view objects from your project and select the key flexfield’s base view object.
8. In the **View Link Accessor Name** field, enter an appropriate name for the view link accessor.

9. Click **Next**. The Source Attributes page appears, as shown in Figure 26-19.

![Figure 26-19 Create Flexfield View Link Wizard — Source Attributes Page for Secondary Mode Key Flexfields](image)

This page is informational only. The key attributes of the source view object will be used to define the view link. The primary key attribute should be listed for this selection.

10. Click **Finish** to go to the Summary page.

    **Note:**
    
    You can skip the Properties page because view link-specific properties are not supported.

11. On the Summary page, review the summary, then click **Finish**.

    The secondary mode key flexfield view link is generated.

### 26.4 Work with Code-Combination Filters for Key Flexfields

A code-combination filter for a key flexfield is a set of query criteria that can be applied to a combinations table to specify a subset of code combinations. After you incorporate a code-combination filter into your application, end users can select key flexfield values in the user interface from the subset produced by the filter.

For example, consider the rows that are listed in Table 26-2. Note that not all columns are shown in the table.
Table 26-2  Example Combinations Table

<table>
<thead>
<tr>
<th>SIN</th>
<th>CCID</th>
<th>Summary_Flag</th>
<th>Enabled_Flag</th>
<th>Segment1_Varchar2</th>
<th>Segment2_Varchar2</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>77</td>
<td>N</td>
<td>Y</td>
<td>8.5X12</td>
<td>YEL</td>
</tr>
<tr>
<td>11</td>
<td>78</td>
<td>N</td>
<td>Y</td>
<td>8.5x14</td>
<td>GRN</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>N</td>
<td>Y</td>
<td>ERGO</td>
<td>NYLON</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>N</td>
<td>Y</td>
<td>HGBAK</td>
<td>LEATHER</td>
</tr>
</tbody>
</table>

You could define a filter with the following conditional logic:

\[
\text{SIN}=14 \text{ and Segment1}_\text{Varchar2}='\text{ERGO}'
\]

When you apply this filter condition to the combinations table, the result listed in Table 26-3 is presented. Note that not all columns are shown in the table.

Table 26-3  Example Filter Result

<table>
<thead>
<tr>
<th>SIN</th>
<th>CCID</th>
<th>Summary_Flag</th>
<th>Enabled_Flag</th>
<th>Segment1_Varchar2</th>
<th>Segment2_Varchar2</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>2</td>
<td>N</td>
<td>Y</td>
<td>ERGO</td>
<td>NYLON</td>
</tr>
</tbody>
</table>

Combination filter conditions for key flexfields are stored in a database column of type XMLType. This column is referred to a a filter-condition column.

There are three types of code-combination filters that you can use in your application — standard code-combination filters, code-combination filters for Oracle Business Intelligence Publisher (Oracle BI Publisher) reports, and cross-validation filters.

### 26.4.1 How to Use Standard Combination Filters

With standard code-combination filters, you determine which key flexfields your end users should be able to filter. Then you define a dedicated filter-condition column in your application database for each filter that you want to include in the application user interface. This column can be defined in an existing reference table. You can also create one or more dedicated tables just to store filter-condition columns.

The filter condition is stored in the filter-condition column as XML. At runtime, the filter condition in the XML is converted to a ViewCriteria object and applied to the key flexfield’s base view object so that when the view object is executed, the filter condition is applied and the filtered query results are produced.

In JDeveloper, you prepare business objects based on the table containing the filter-condition column, then you associate a combination filter view object attribute with the key flexfield. You can associate zero, one, or many combination filters with a given key flexfield, but only one flexfield can be addressed by a given filter.

To make the code-combination filter accessible to application implementors or administrators, add a code-combination filter UI component to an application page. Each row contains a different filter definition that can be applied to the associated key flexfield. The implementors or administrators will be responsible for populating the table with filter criteria using a provided utility.
Code-combination filters for key flexfields are supported by the XML schema FndFilter.xsd. This XML schema binds the filter XML that is defined. This schema is registered with the FUSION database schema at the following URI:

http://www.oracle.com/apps/fnd/appcore/filter/FndFilter.xsd

The XML schema is registered to the database as BINARY_XML.

You can test the filter definitions by inserting predefined XML filter criteria into the filter-condition column.

**Note:**

A PL/SQL API is provided so that you can apply filters to your SQL statements as WHERE clause conditions rather than applying them to the user interface. For more information, see How to Apply Code-Combination Filters Using the PL/SQL Filter APIs.

Code-combination filters are removed from an application by removing their accessors.

### 26.4.2 How to Use Code-Combination Filters for Oracle BI Publisher Reports

The Applications Core key flexfield filter repository enables Oracle Fusion Applications developers to include selected key flexfield segments as available parameters in an Oracle BI Publisher report submission user interface. The filter-repository mechanism translates report parameters for those segments into code-combination filter criteria, which are then translated into SQL for inclusion in the report. You accomplish this by first creating a flexfield filter view object for the public entity object FndKfEssFiltersPEO to access a provided common filter repository table, and then by adding to the report submission page a filter UI component that is based on the filter view object.

When the report job is submitted, the flexfield filter XML definition produced by the filter input criteria is saved to the filter repository. Oracle Enterprise Scheduler Service start the reporting job with the report parameters including the filter key. The filter key is passed to the flexfield lexical API, which returns the filter criteria as a SQL WHERE clause, which Oracle BI Publisher integrates into the SQL statement for its report.

To incorporate code-combination filters for Oracle BI Publisher reports into a maintenance user interface, you follow much of the same process as you would to implement standard filters. The code-combination filter procedures that follow note which procedures do not apply to these types of filters.

The kff_filter_purge(...) procedure from the fnd_flex_xml_publisher/apis PL/SQL package enables you to remove unused filters from the filter repository.

### 26.4.3 How to Use Cross-Validation Filters

A cross-validation rule applies a pair of filters to new code combinations that are proposed for a key flexfield by administrators or end users, when you have enabled the rule to work with maintenance mode or dynamic combination insertion.

After enabling cross-validation for a key flexfield at registration time, you must build a maintenance user interface that administrators can use to maintain the
implementation-specific filters that make up each rule. All filter combinations that an administrator defines for a given key flexfield are applied automatically to cross validate new code combinations as they are entered.

**Note:**

Cross-validation rule criteria should generally be created and modified only by application implementors and administrators. For end users, these rules automatically validate new code combinations in the same way that value sets automatically validate new segment values.

To incorporate cross-validation filters, you follow much of the same process as you would to implement standard filters. The code-combination filter procedures that follow note which procedures do not apply to these types of filters.

For more information about implementing cross-validation rules, see How to Implement Cross-Validation Rules and Custom Validation.

### 26.4.4 How to Prepare the Database for Standard Code-Combination Filters

A database column of type `XMLType` is required to store filter data in your database. This column is referred to as the filter-condition column. For standard code-combination filters, you must define a filter-condition column for the filter data in your database before you can associate code-combination filters with key flexfields in your application.

**Note:**

If you are implementing code-combination filters to support cross-validation rules, the required filter-condition columns already exist in the `FND_KF_CROSS_VAL_RULES` repository table. If you are implementing code-combination filters for use in the key flexfield filter repository for Oracle BI Publisher reports, these columns exist in the repository.

To prepare a standard code-combination filter for modeling:

1. Select an existing table to contain your filter, or create a new one.

   To create a table called, for example, `FND_MYFILTER_KFF1`, execute the following script:
   
   ```sql
   Create table FND_MYFILTER_KFF1 (ID Number primary key, Info Varchar2(1000));
   ```

2. Use the following `Alter` script to add a filter-condition column (for example, `Filter`) of type `XMLType` to your table:
   
   ```sql
   Alter table FND_MYFILTER_KFF1 add Filter xmltype
   XMLType column Filter
   Store as BINARY XML
   XMLSCHEMA "http://www.oracle.com/apps/fnd/applcore/flex/kff/FndFilter.xsd"
   ELEMENT "KeyFlexCodeCombinationFilter",
   ```
Note:
Your new filter-condition column must be configured as nullable.
This script is necessary because the Database Schema Deployment framework
does not support the XMLType data type.

26.4.5 How to Add Code-Combination Filters to Your Application

To add code-combination filters to your application, you complete the following tasks:

1. For standard filters only, create an entity object for the table containing the filter-
   condition column.
2. Create a view object for the filter entity object.
3. Associate the code-combination filters with key flexfields.
4. Configure, deploy, and test the code-combination filters.

26.4.5.1 Create a Filter Entity Object for a Standard Filter

For standard filters, you must create a filter-specific entity object for the table
containing the filter-condition column.

Note:
You do not need to create a filter view object if you are implementing one of
the following types of filters:

- If you are implementing code-combination filters for use in the key
  flexfield filter repository for Oracle BI Publisher reports, use the existing
  public entity object
  oracle.apps.fnd.applcore.flex.kff.model.publicEntity.F
  ndEssFiltersPEO, which became available when you added the
  Applications Core library to your data model project.
- If you are implementing code-combination filters to support cross-
  validation rules, use the provided configured entity object:
  oracle.apps.fnd.applcore.flex.kff.model.entity.KeyFlexfieldCrossValidatio
  nRuleEO

Before you begin:

1. Define a database column of XMLType to store the filter as described in How to
   Prepare the Database for Standard Code-Combination Filters.
2. To use code-combination filters, you must first have completed the Create
   Flexfield Business Components wizard for at least one key flexfield, so that your
   project contains one or more key flexfield business components.

To create the filter entity object:
1. Create an entity object (for example, Kff1Fltr1EO) for the table containing your filter-condition column.

2. Open the entity object, and, in the overview editor, click the General navigation tab.

3. Expand the Custom Properties section and add the FND_FILTER property with a value of Y. This property enables the base classes (OAEntityImpl) to recognize that the entity object contains a filter attribute.

4. Because the column type of the filter attribute is XMLType, which is not natively supported by ADF Business Components, you must edit the attribute to make it a transient attribute that is computed using a SQL expression.

   Click the Attributes navigation tab, select the filter attribute, and click the Edit icon to open the Edit Attribute dialog.

5. In the Entity Attribute dialog, click the Entity Attribute node and change the attribute from a persistent type to a calculated type, as shown in Figure 26-20.

   a. Specify the ClobDomain type.

   b. Select Derived from SQL Expression.

   c. Enter an Expression such as the following expression:

      \[ \text{Kff1Fltr1EO.filter.getClobVal()} \]

---

**Note:**

The GetClobVal() method is needed to manage the XMLType column in the database because ADF Business Components currently does not support the XMLType data type natively.
6. Click the Custom Properties node, as shown in Figure 26-21.

7. Add the custom filter properties that are listed in Table 26-4 to the filter attribute.
### Table 26-4  Custom Filter Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FND_FILTER</td>
<td>Y</td>
<td>Indicates that the entity attribute is a filter attribute.</td>
</tr>
<tr>
<td>FND_FILTER_TABLE</td>
<td>table-name</td>
<td>Indicates the name of the underlying table on which this filter attribute is based.</td>
</tr>
<tr>
<td>FND_FILTER_COL</td>
<td>column-name</td>
<td>Indicates the name of the column on which this attribute is based in the filter table. This is needed because the entity object could be based on a database view.</td>
</tr>
<tr>
<td>FND_FILTER_TABLE_COL_PK_n</td>
<td>primary-key-column-id</td>
<td>Indicates the primary key column of the underlying filter table. If the table has a composite primary key (for example: ID1, ID2), you must add an entry for each key. For example: FND_FILTER_TABLE_COL_PK1=ID1 FND_FILTER_TABLE_COL_PK2=ID2</td>
</tr>
<tr>
<td>FND_FILTER_TABLE_ATTR_PK_n</td>
<td>view-object-attribute-name</td>
<td>Indicates the name of the view object attribute that corresponds to the attribute in the entity object that represents the filter table primary key. If the view object has attributes that correspond to multiple entity object primary key attributes, you must add an entry for each key. For example: FND_FILTER_TABLE_ATTR_PK1=I D1 FND_FILTER_TABLE_ATTR_PK2=I D2</td>
</tr>
</tbody>
</table>

26.4.5.2 Create a Filter View Object

You must create a filter view object (for example, Kff1Fltr1VO) for the filter entity object. How you create the view object depends on how you will use the filter:
• If you are implementing a standard code-combination filter, then create the view object for the entity object that you created in Creating a Filter Entity Object for a Standard Filter.

• If you are implementing the filter for use in the key flexfield filter repository for Oracle BI Publisher reports, then create the view object for the public entity object:

  oracle.apps.fnd.applcore.flex.kff.model.publicEntity.FndKfEssFiltersPEO

• If you are implementing the filter to support cross-validation rules, then create the view object for the provided cross-validation entity object:

  oracle.apps.fnd.applcore.flex.kff.model.entity.KeyFlexfieldCrossValidationRuleEO

  In the view object for the cross-validation rules, define view criteria to set the APPLICATION_ID and KEY_FLEXFIELD_CODE attributes to static values for your application and key flexfield.

26.4.5.3 Associate Code-Combination Filters with Key Flexfields

You use the Create Flexfield Filter wizard to create a view accessor from the filter view object's code-combination filter attribute to a key flexfield view object definition.

---

**Note:**

If you are implementing filters to support cross-validation rules, you must complete this procedure twice — once for the condition filter attribute and once for the validation filter attribute.

---

**To associate a code-combination filter with a key flexfield:**

1. In the Application Navigator, right-click the project and choose New.

2. In the New Gallery, expand Business Tier, select ADF Business Components and then Flexfield Filter, and click OK.

3. In the Filter Accessor dialog, expand the available view objects in your current project on the left-hand list and select the view object attribute that corresponds to the XML Filter column as shown in Figure 26-22.
4. Expand the available flexfields in your current project on the right-hand list and select a key flexfield to be filtered.

5. Enter a name for the filter accessor (with no spaces), then click **OK**.

### 26.4.5.4 Configure, Deploy, and Test Code-Combination Filters

The final task is to configure the view object, add it to a new application module for the filter, and test it.

**To configure, deploy, and test a code-combination filter:**

1. Open the filter view object and click the **General** navigation tab.
   
   This property enables the base classes (OAViewRowImpl) to recognize that the view object row contains a filter attribute.

2. Expand the **Custom Properties** section and add the property **FND_FILTER** with the **Value** set to **Y**.

3. Click the **Attributes** navigation tab and select the filter attribute.

4. Expand the **Custom Properties** section and add the property **FND_ACFF_SIN** for the selected filter attribute. Set the **Value** to the structure instance number (SIN).
   
   This property indicates the view object's SIN attribute that is associated with this filter attribute.
5. Create an application module for the filter. In the Data Model page, move the filter view object to the **Data Model** list. In the Application Modules page, move the flexfield application module, which was created when you created the flexfield business component, to the **Selected** list.

**Tip:**
You also can add the flexfield application module from the **Application Module Instances** section in the Data Model navigation tab for the application module.

6. Run the Business Component Browser to ensure that all attributes appear.

### 26.4.6 How to Employ Code-Combination Filters on an Application Page

You must add the filter view accessors that you created to an application page. This procedure applies to conventional key flexfield filters as well as to filters that you are implementing for use in the filter repository.

#### 26.4.6.1 Add Your Key Flexfield Filter to an Application Page

You add a key flexfield filter to an application page by dropping the filter view object onto the page and modifying the XML code for the filter component.

---

**Note:**
This procedure is also used to produce a user interface for defining and maintaining cross-validation rules. Complete the procedure twice — once for the accessor of the condition filter and once for the accessor of the validation filter. Both filters can be exposed on the same page.

---

**Before you begin:**

- Ensure that the Applications Core (ViewController) tag library has been added to the user interface project, as described in [Adding the Applications Core Tag Library to Your User Interface Project](#).

- Create the view object, view accessor, and application module for the code-combination filter as described in [How to Add Code-Combination Filters to Your Application](#).

**To add your key flexfield filter to an application page:**

1. In your project, create a new JSPX page.

2. Drag and drop the view object that contains your filter from the Data Controls panel onto the page as an ADF Form or an ADF Table. **Figure 26-23** shows the filter view object dropped onto the page as a form.
3. Ensure the `CreateInsert` and `Commit` actions are included on the page.

   **Note:**
   These actions enable dynamic creation of new filter definitions at runtime; they also enable you to insert new records into the filter repository.

4. If you dropped the view object onto the page as an ADF Table, select the filter column and, in the Property Inspector, set `Sortable` to `false`.

   **Note:**
   The sorting of filter columns is not supported.

5. If you dropped the view object onto the page as an ADF Form, select the filter component and, in the Property Inspector, enter the name of the flexfield in the `Label` field. This name is used in the title of the filter popup dialog.

6. By default, end users can choose to match on *all* conditions or *any* condition, as shown in Figure 26-24 and Figure 26-25. If you want to restrict the filter to match on all conditions only, add the `RestrictConjunctionToAND` property and set it to `true`, as shown in Example 26-13 and Example 26-14.

   When this property is set to `true`, the Filter dialog box does not display the Match options, as shown in Figure 26-26. All of the conditions for the same segment are joined with an OR operator and then combined with the conditions for other segments using an AND operator, as shown in Example 26-15.
Work with Code-Combination Filters for Key Flexfields

Figure 26-24  Form-Based Code-Combination Filter User Interface

Figure 26-25  Table-Based Code-Combination Filter User Interface

Figure 26-26  Filter Dialog When RestrictConjunctionToAND is Set to true

Example 26-13  Modified Form-Based Filter Code

```
<fnf:keyFlexFilter value="#{bindings.Kff1Fltr1_1Iterator}" accessor="kff1"
     label="#{bindings.Filter.hints.label}" id="kff1"
     restrictConjunctionToAND="true"/>
```
Example 26-14  Modified Table-Based Filter Code

<af:column sortProperty="Filter" sortable="false"
        headerText="#{bindings.Kff1Fltr1_1.hints.Filter.label}"
        id="c2">
  <fnd:keyFlexFilter value="#{bindings.Kff1Fltr1_1Iterator}"
        accessor="kff1" id="kff1"
        restrictConjunctionToAND="true"/>
</af:column>

Example 26-15  Joining of Segments When restrictConjunctionToAND is true

(SEGMENT1 = 'A' or SEGMENT = 'B') AND (SEGMENT2 = 10 OR SEGMENT2 = 20)

26.4.6.2 What Happens When You Add a Filter-Repository Filter to an Application Page

When you add a filter based on the public entity object FndKfEssFiltersPEO to your application page as an ADF Form, the resulting Oracle BI Publisher report submission user interface appears as shown in Figure 26-27.

Figure 26-27  Form-Based Report Submission Code-Combination Filter UI

When you add a filter-repository filter to the application page as an ADF Table, the report submission user interface appears as shown in Figure 26-28.

Figure 26-28  Table-Based Report Submission Code-Combination Filter UI
When you click **CreateInsert**, a new row is added that includes the filter XML and other required input, along with the default values for some of the columns. Your application must provide defaults for the columns as described in **Table 26-5**.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyFlexfieldCode</td>
<td>The code identifying the key flexfield to which this filter will be applicable. This is a read-only value.</td>
</tr>
<tr>
<td>StructureInstanceNumber</td>
<td>This is the SIN, the discriminator attribute for the key flexfield that is used in the key flexfield filter. While creating a new filter definition or submitting a new job, a valid value should be the default for this attribute at the view object level. The SIN is required for capturing the filter XML. This is a read-only attribute.</td>
</tr>
<tr>
<td>DataSetNumber</td>
<td>The data set number (DSN) is a secondary discriminator to the SIN. If the key flexfield is data set-enabled, a valid DSN value should be the default in the filter view object. This is a read-only attribute.</td>
</tr>
<tr>
<td>FilterId</td>
<td>The FilterId is the primary key attribute and is a unique identifier for each filter that is inserted in the filter repository. This value can be generated using a sequence or other methods for generating unique identifiers.</td>
</tr>
<tr>
<td>ApplicationShortName</td>
<td>This is the application short name of the application with which the flexfield filter is associated. You should set the default value to be the application with which your key flexfield is associated. For example, if you are using flexfield KFF1, which is associated with the Application Object Library application, your filter repository should set the default value for <strong>ApplicationShortName</strong> to be <strong>FND</strong>. This is a read-only attribute.</td>
</tr>
<tr>
<td>Filter</td>
<td>This is the XML attribute containing the <em>WHERE</em> condition that is set for a particular FilterId. The <em>WHERE</em> condition has to be populated by using the filter user interface. Depending on the SIN, the filter user interface displays the related segments for a particular key flexfield structure. Various conditions for each of the segments can be applied to generate the <em>WHERE</em> condition.</td>
</tr>
</tbody>
</table>

When you click **Commit**, the new row is inserted in the FND_KF_ESS_FILTERS database table.

### 26.4.7 How to Create Code-Combination Filter Definitions for Testing

For testing, you can use **INSERT** scripts to insert predefined XML filter criteria into the filter-condition column of your product table.

**Note:**

You can insert this data at any time after the filter-condition column has been added to the product table, and before the filter is invoked.
Use the following form to build an INSERT script:

```
Insert into filtercolumnname values(indexnum, 'filterconditionname',
XMLType('filter_xml_code'))
```

The operators supported for code-combination filters are the operators supported in
the Query panel. This includes the following data types and their operators:

- **STRING data type**: EQUALTO, NOTEQUALTO, CONTAINS, DOESNOTCONTAIN,
  LIKE, STARTSWITH, ENDSWITH, ISNULL, ISNOTNULL

- **NUMBER data type**: EQUALTO, NOTEQUALTO, NULL, ISNOTNULL, GREATER_THAN,
  LESS_THAN, GREATERTHANEQUALTO, LESSTHANEQUALTO, BETWEEN,
  NOTBETWEEN

- **DATE data type**: ISNULL, ISNOTNULL

You can also use the following hierarchical operators to query tree structures in your
filter: IS_CHILD_OF, IS_DESCENDANT_OF, IS_LAST_DESCENDANT_OF,
IS_PARENT_OF, IS_ANCESTOR_OF, IS_FIRST_ANCESTOR_OF, IS_SIBLING_OF.

For more information about trees, see Organizing Hierarchical Data with Tree
Structures.

**Example 26-16** shows some example scripts. The first one inserts a filter condition that
selects for `SEGMENT1_VARCHAR2 = 'Value04'`, and the second one selects for the
inequality `SEGMENT1_VARCHAR2 != 'Value02'`.

**Note:**

You might want to test these scripts to ensure that the database is, in fact,
performing schema validation on the XML document, by attempting to insert
XML that does not conform to this schema.

**Example 26-16  Scripts for Inserting Filter Conditions into the Application Database**

Example of EQUALTO filter:

```
Insert into KFF1_FLTR values(
1, 'EqualToFilter',
XMLType('<!-- xml version ="1.0" encoding ="UTF-8" -->
<FndFilter xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
FndFilter.xsd">
<KeyFlexFilter>
/*/keyFlexfieldCode>KFF1</keyFlexfieldCode>
<structureInstanceCode>VS_IND_CHR_ON_CHR</structureInstanceCode>
<applicationShortName>FND</applicationShortName>
<filterCriteriaRow>
<filterCriteriaItem>
<attributeName>_L10</attributeName>
<columnName>SEGMENT1_VARCHAR2</columnName>
<operator>EQUALTO</operator>
<conjunction>AND</conjunction>
<valueDataType>STRING</valueDataType>
<value>Value04</value>
<properties>
<property>
<name>TestProp</name>
<value>ValueProp</value>
```
Example of \texttt{NOTEQUALTO} filter:

\begin{verbatim}
Insert into KFF1_FLTR values(
  2, 'NotEqualToFilter',
  XMLType('<?xml version ="1.0" encoding ="UTF-8"?>
  <FndFilter xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    FndFilter.xsd">
    <KeyFlexFilter>
      <keyFlexfieldCode>KFF1</keyFlexfieldCode>
      <structureInstanceCode>VS_IND_chr_ON_chr</structureInstanceCode>
      <applicationShortName>FND</applicationShortName>
      <properties>
        <property>
          <name>TestKff</name>
          <value>Valkff</value>
        </property>
      </properties>
    </KeyFlexFilter>
    <filterCriteriaRow>
      <filterCriteriaItem>
        <attributeName>_L10</attributeName>
        <columnName>SEGMENT1_VARCHAR2</columnName>
        <operator>NOTEQUALTO</operator>
        <conjunction>AND</conjunction>
        <valueDataType>STRING</valueDataType>
        <value>Value02</value>
        <properties>
          <property>
            <name>TestItemName</name>
            <value>ValItem</value>
          </property>
        </properties>
      </filterCriteriaItem>
      <conjunction>AND</conjunction>
      <properties>
        <property>
          <name>TestRowName</name>
          <value>ValRow</value>
        </property>
      </properties>
    </filterCriteriaRow>
  </FndFilter>'));
\end{verbatim}

\subsection{26.4.8 How to Apply Code-Combination Filters Using the PL/SQL Filter APIs}

You can take advantage of code-combination filters (including filter-repository filters) without using them in your application user interface. You use the \texttt{WHERE} clause API
for standard and cross-validation combination filters, and you use the XML API for filter repository filters.

### 26.4.8.1 Apply Standard Filters Using the WHERE Clause API

Applications Core provides a PL/SQL API for filtering at the back end. This API takes a filter condition as an input parameter in `XMLType` format, converts it to a SQL `WHERE` clause, and provides the clause as an output parameter for the segments upon which the filter condition has been defined. You use this API to integrate the `WHERE` clause into your SQL statements to include the filter conditions within your SQL scripts.

The PL/SQL combination filter `WHERE` clause API is based on the signature shown in **Example 26-17**.

The bind values are defined as shown in **Example 26-18**.

**Example 26-19, Example 26-20, and Example 26-21** demonstrate how to use the `WHERE` clause API for an `EQUALTO` condition, a `BETWEEN` condition, and multiple conditions.

**Example 26-17  WHERE Clause Signature**

```sql
FND_KF_COMBINATION_FILTER_API.BuildWhereClause(
    filter            IN XMLType,
    tableAlias        IN Varchar2,
    bindPrefix        IN Varchar2,
    sin               OUT Number,
    bindValues        OUT NOCOPY BIND_VAL_TAB,
    filterWhereClause OUT NOCOPY Varchar2);
```

/**
-- This procedure computes the WHERE clause for the filter
-- and provides it in the filterWhereClause parameter
--
-- IN Params
--     filter      XMLType  - Filter to be converted to SQL clause
--     tableAlias  Varchar2 - Alias table name to be used in SQL clause
--     bindPrefix  Varchar2 - Bind Prefix
-- OUT Params
--     sin         Number    - Structure Instance Number
--     bindValues  BIND_VAL_TAB - List of Bind Values
--     filterWhereClause Varchar2 - WHERE clause
--
**************************************************************************/

**Example 26-18  Bind Values Definition**

```sql
create or replace PACKAGE FND_KF_COMBINATION_FILTER_API AS

VARCHAR_TYPE CONSTANT Varchar2(20) := 'VARCHAR';
NUMBER_TYPE CONSTANT Varchar2(20) := 'NUMBER';
DATE_TYPE CONSTANT Varchar2(20) := 'DATE';

TYPE BIND_VALUE IS RECORD(
    NAME            Varchar2(30),
    TYPE            Varchar2(20),
    VALUEVARCHAR2  Varchar2(32767),
    VALUE_NUMBER   Number,
    VALUE_DATE     Date);

TYPE BIND_VAL_TAB IS TABLE OF BIND_VALUE INDEX BY BINARY_INTEGER;
```

Use the Advanced Features of Key Flexfields 26-59
Example 26-19   Using the WHERE Clause API for an EQUALTO Condition

Suppose that a filter condition has been defined in a combinations table as follows:

- Combinations table = FND_KF_TEST_CCT1
- Filter column = SEGMENT1_VARCHAR2
- Filter condition = 123

You would call the filter API as follows:

```java
FND_KF_COMBINATION_FILTER_API.BuildWhereClause(
    filter=>v_filter,
    tableAlias => 'FKFF1',
    bindPrefix => 'BND',
    sin => v_sin,
    bindValues => v_bind,
    filterWhereClause => v_query);
```

Use the `tableAlias` value in WHERE clauses to represent the combinations table name. In this example, `FND_KF_TEST_CCT1.SEGMENT1_VARCHAR2` should be entered as `FKFF1.SEGMENT1_VARCHAR2`.

Similarly, use the `bindPrefix` value as a prefix when referencing individual bind values, for example, `:BND1`, `:BND2`, or `:BND3`.

When invoked, the filter API in this example might produce the following values for its output parameters:

- `filterWhereClause` - `FKFF1.SEGMENT1_VARCHAR2 = :BND1`
- `sin` - 12
- `bindValues` - `bindValues(1).NAME = BND1`
  - `bindValues(1).TYPE = VARCHAR2`
  - `bindValues(1).VALUE_VARCHAR2 = 123`

With this output, you can assemble the following WHERE clause for an EQUALTO filter condition:

```sql
select code_combination_id,
       Segment1_VARCHAR2
from FND_KF_TEST_CCT1 FKFF1
where FKFF1.structure_instance_number=12
  and FKFF1.SEGMENT1_VARCHAR2=123
```

Example 26-20   Using the WHERE Clause API for a BETWEEN Condition

The following shows an example of a BETWEEN operator used as part of a filter expression of the form "attribute BETWEEN value1 AND value2".

```xml
<?xml version ="1.0" encoding ="UTF-8"?>
<KeyFlexCodeCombinationFilter
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=
"http://www.oracle.com/apps/fnd/applcore/flex/kff/KeyFlexCodeCombinationFilter.xsd">
  <keyFlexfieldCode>KFF1</keyFlexfieldCode>
  <structureInstanceCode>VS_FRM_NUM_ON_CHR</structureInstanceCode>
  <applicationShortName>FND</applicationShortName>
  <filterCriteriaRow>
    <filterCriteriaItem>
```

26-60 Oracle Fusion Applications Developer's Guide
The filter expression captured in the preceding XML resolves to the following:

```
SEGMENT1VARCHAR2 BETWEEN -500 AND 1000
```

You would call the filter API as follows:

```
FND_KF_COMBINATION_FILTER_API.BuildWhereClause(
    filter=>v_filter,
    tableAlias =>'FKFF1',
    bindPrefix =>'BND',
    sin => v_sin,
    bindValues => v_bind,
    filterWhereClause => v_query);
```

When invoked, the filter API in this example might produce the following values for its output parameters:

```
filterWhereClause - FKFF1.SEGMENT1VARCHAR2 = :BND1
sin - 12
bindValues - bindValues(1).NAME = BND1
- bindValues(1).TYPE = VARCHAR2
- bindValues(1).VALUEVARCHAR2 = -500
- bindValues(2).NAME = BND2
- bindValues(2).TYPE = VARCHAR2
- bindValues(2).VALUEVARCHAR2 = 1000
```

With this output, you can assemble the following WHERE clause for a BETWEEN filter condition:

```
select code_combination_id,
    Segment1VARCHAR2
from FND_KF_TEST_CCT1 FKFF1
where FKFF1.structure_instance_number=12
and FKFF1.SEGMENT1VARCHAR2 BETWEEN -500 AND 1000
```

### Example 26-21 Using the WHERE Clause API for Multiple Conditions

The following shows an example of multiple operators used as part of a filter expression of the form "attribute1 EQUALTO value1 AND attribute2 EQUALTO value2."

```xml
<?xml version ="1.0" encoding ="UTF-8"?>
<KeyFlexCodeCombinationFilter
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
KeyFlexCodeCombinationFilter.xsd">
    <keyFlexfieldCode>KFF1</keyFlexfieldCode>
    <structureInstanceCode>VS_FRM_NUM_ON_CHR</structureInstanceCode>
    <applicationShortName>FND</applicationShortName>
</KeyFlexCodeCombinationFilter>
```
The filter expression captured in the preceding XML resolves to the following:

SEGMENT1_VARCHAR2 = 123456 and SEGMENT2_VARCHAR2 = 123.45

You would call the filter API as follows:

```java
FND_KF_COMBINATION_FILTER_API.BuildWhereClause(
    filter=>v_filter,
    tableAlias => 'FKFF1',
    bindPrefix => 'BND',
    sin => v_sin,
    bindValues => v_bind,
    filterWhereClause => v_query);
```

When invoked, the filter API in this example might produce the following values for its output parameters:

- `filterWhereClause`:
  - FKFF1.SEGMENT1_VARCHAR2 = :BND1
  - FKFF1.SEGMENT2_VARCHAR2 = :BND2
- `sin`:
  - 12
- `bindValues`:
  - bindValues(1).NAME = BND1
  - bindValues(1).TYPE = VARCHAR2
  - bindValues(1).VALUE_VARCHAR2 = 123456
  - bindValues(2).NAME = BND2
  - bindValues(2).TYPE = VARCHAR2
  - bindValues(2).VALUE_VARCHAR2 = 123.45

With this output, you can assemble the following `WHERE` clause for a BETWEEN filter condition:

```sql
select code_combination_id,
       Segment1_VARCHAR2, Segment2_VARCHAR2
from FND_KF_TEST_CCT1 FKFF1
where FKFF1.structure_instance_number=12
  and FKFF1.SEGMENT1_VARCHAR2 = 123456
  and FKFF1.SEGMENT2_VARCHAR2 = 123.45
```
26.4.8.2 Apply Repository Filters for Oracle Enterprise Scheduler Service

The `kff_filter` PL/SQL procedure in the `fnd_flex_xml_publisher_apis.pkb` package is the public procedure for processing key flexfield repository filter lexicals. The signature is shown in Example 26-22.

Example 26-23 demonstrates how to use this API to obtain the `WHERE` clause and bind variable information for a filter in the filter repository.

**Example 26-22  `kff_filter` Signature**

```sql
/* PUBLIC PROCEDURE kff_filter EXPOSED FOR THIS PACKAGE */

PROCEDURE kff_filter
(p_lexical_name                 IN VARCHAR2,
 p_application_short_name       IN fnd_application.application_short_name%TYPE,
 p_key_flexfield_code           IN fnd_kf_flexfields_b.key_flexfield_code%TYPE,
 p_filter_id                    IN NUMBER,
 p_code_combination_table_alias IN VARCHAR2,
 x_where_expression             OUT nocopy VARCHAR2,
 x_numof_bind_variables         OUT nocopy NUMBER,
 x_bind_variables               OUT nocopy bind_variables);
```

**Example 26-23  Using the Filter Repository API**

```sql
REM dbdrv: none
SET SERVEROUTPUT ON
WHENEVER SQLERROR CONTINUE
DECLARE
  l_tableAlias VARCHAR2(30);
  l_applicationShortName fnd_application.application_short_name%TYPE;
  l_keyFlexfieldCode fnd_kf_flexfields_b.key_flexfield_code%TYPE;
  l_filterWhereClause         VARCHAR2(32767);
  l_filterId                  NUMBER;
  l_filterName                VARCHAR2(32);
  l_bindVariables fnd_flex_xml_publisher_apis.bind_variables;
  l_numOfBindVariables NUMBER;
CURSOR c_filter_id
IS
  SELECT filter_id FROM fnd_kf_ess_filters;
BEGIN
  l_filterName           := 'DefaultFilter';
  l_tableAlias           := 'DefaultTable';
  l_keyFlexfieldCode     := 'KFF1';
  l_applicationShortName := 'FND';
  DBMS_OUTPUT.PUT_LINE('kff_filter');
  FOR filter_id IN c_filter_id
    LOOP
    fnd_flex_xml_publisher_apis.kff_filter(p_lexical_name=>l_filterName,
                                         p_application_short_name=>l_applicationShortName,
                                         p_key_flexfield_code=>l_keyFlexfieldCode,
                                         p_filter_id=>filter_id.filter_id,
                                         p_code_combination_table_alias=>l_tableAlias,
                                         x_where_expression=>l_filterWhereClause,
                                         x_numof_bind_variables=>l_numOfBindVariables,
                                         x_bind_variables=>l_bindVariables);
    DBMS_OUTPUT.PUT_LINE('filter Id: ' || filter_id.filter_id);
    DBMS_OUTPUT.PUT_LINE('filter Where Clause: ' || l_filterWhereClause);
  END LOOP;
END;
```

Use the Advanced Features of Key Flexfields 26-63
26.4.9 How to Use an API to Apply XML Filters

Currently the only way data can be inserted into our Filter columns is through our Filter APIs. However some of the teams who have to insert thousands of the CVR rules are finding it a bit time consuming.

You can use a filter API to take in a list of conditions and generate filter XML to be inserted into the flexfield filter columns. The code that calls the API must pass some basic information about the key flexfield for which the Filter XML is to be generated. This information is required to correctly identify the subtype flexfield view object, which is then used to get the list of attributes.

Use the FlexFilterCriteriaItemBean class to pass the list of conditions for which the filter XML must be generated. FlexFilterCriteriaItemBean includes three attributes which are used to construct individual filter criteria.

This API is useful when having to process large amounts of data.

The following sample shows the API signature for FlexFilterCriteriaItemBean.

The following sample illustrates the use of the FlexFilterCriteriaItemBean class.

The code shown in the previous sample generates XML which produces the following condition:

\[ (_{L10}= 1 \text{ OR } _{L10} \text{ BETWEEN } 50 \text{ and } 100) \text{ AND } (_{L10}.\text{NUMONLY} \text{ BETWEEN } 2 \text{ and } 40) \text{ AND } (_{L10}.\text{ZEROFILL IS DESCENDENT OF 199}) \]

**Example 26-24  API Signature for FlexFilterCriteriaItemBean**

```java
/**
 * Creates FilterXML from a given set of conditions passed in the form of a list
 * of FilterCriteriaItem.
 * @param kffVO is the key flexfield base view object.
 * @param criteriaItems is the list of conditions which is to be converted to
 * @filter XML.
 * @param sin SIN of the subtype FlexVO for which the XML is being generated
 * @param dsn DSN if it exists. Else pass null.
 * @return Filter XML when all the arguments are valid.
 * @exception JboException
 * @exception IllegalArgumentException
 */
public static ClobDomain createFilterXML(KFFViewObjectImpl kffVO,
                                          List<KeyFlexFilterCriteriaItem> criteriaItems,
                                          Long sin, Long dsn)
```

```java
/**
 * Data Structure used to capture individual an FilterCriteriaItem.
 * This data structure can be used in the following ways to populate various
 * conditions.
 * KeyFlexFilterCriteriaItem item1 = new KeyFlexFilterCriteriaItem();
 * item1.setSegmentCode("L10");
 * item1.setOperator("=");
 * String[] values = {"1"};
 * item1.setValues(values);
 * This is equivalent to - 'ColumnName(L10) = 1'
 */
```

Work with Code-Combination Filters for Key Flexfields
KeyFlexFilterCriteriaItem item2 = new KeyFlexFilterCriteriaItem();
item2.setSegmentCode("L10_NUMONLY");
item2.setOperator("BETWEEN");
String[] values2 = {"99", "199"};
item2.setValues(values2);
This is equivalent to - 'ColumnName(L10_NUMONLY) BETWEEN 99 and 199'

KeyFlexFilterCriteriaItem item3 = new KeyFlexFilterCriteriaItem();
item3.setSegmentCode("L10_ZEROFILL");
item3.setOperator("IS_DESCENDENT_OF");
String[] values3 = {"199"};
item3.setValues(values3);
This is equivalent to - ' ColumnName(L10_ZEROFILL) IS_DESCENDENT_OF 199'

Rules to follow while creating the KeyFlexFilterCriteriaItems:
* For operators 'ISBLANK' and 'ISNOTBLANK' variable 'values' should be null.
* For operators 'BETWEEN' and 'NOTBETWEEN' variable 'values' should have 2 values.
* For other operators, variable 'values' should have only 1 value.
* This class will not validate any of the inputs. Passing the list of KeyFlexFilterCriteriaItem to createFilterXML results in an IllegalArgumentException if the rules are not followed.
* The 'conjunction' variable is used while grouping together the same 'segmentCode'.
* Always use AND for different 'segmentCode' items.
* Always use OR for the same 'segmentCode' items.

public class KeyFlexFilterCriteriaItem
{
    private String segmentCode;
    private String operator;
    private String[] values;
}

Example 26-25  Using FlexFilterCriteriaItemBean

Kff1Ref1AMImpl flexAM =
    (Kff1Ref1AMImpl)
OAUtility.createRootApplicationModule("oracle.apps.fnd.applcore.flex.test.model.applicationModule.Kff1Ref1AM", "Kff1Ref1AMLocal");
KFFViewObjectImpl kffVO = (KFFViewObjectImpl)
(flexAM.findViewObject("Kff1Instance"));

ArrayList<KeyFlexFilterCriteriaItem> r =
    new ArrayList<KeyFlexFilterCriteriaItem>(){
    KeyFlexFilterCriteriaItem item1 = new KeyFlexFilterCriteriaItem();
    item1.setSegmentCode("L10");
    item1.setOperator("=");
    item1.setValues("1");
    r.add(item1);
    KeyFlexFilterCriteriaItem item3 = new KeyFlexFilterCriteriaItem();
    item3.setSegmentCode("L10_NUMONLY");
    item3.setOperator("BETWEEN");
item3.setValues("50", "100");
r.add(item3);
KeyFlexFilterCriteriaItem item2 = new KeyFlexFilterCriteriaItem();
item2.setSegmentCode("L10");
item2.setOperator("BETWEEN");
item2.setValues("2", "40");
r.add(item2);
KeyFlexFilterCriteriaItem item4 = new KeyFlexFilterCriteriaItem();
item4.setSegmentCode("L10_ZEROFILL");
item4.setOperator("IS_DESCENDENT_OF");
item4.setValues("199");
r.add(item4);

ClobDomain filterClob = FilterManager.createFilterXML(kffVO,r,5L, null);

26.4.10 How to Remove Code-Combination Filters from Your Application

To remove a code-combination filter, you remove the accessor that was previously created to associate the filter with a particular key flexfield.

In your project, right-click the view object that contains the filter and select Remove Flexfield Filters from the menu.

If the filter view object has more than one filter attribute with an accessor defined, then you will be presented with a list of those filter accessors. Select the one that you want to remove.

26.4.11 How to Remove Filters from the Filter Repository

The filter XML is stored in the FND_KF_ESS_FILTERS table. The number of rows in a filter repository can become large. You use the kff_filter_purge(...) procedure from the fnd_flex_xml_publisher_apis PL/SQL package to purge unused filters from the filter repository. This procedure takes the filter’s ID, as shown in Example 26-26.

Example 26-26   Removing a Filter from the Filter Repository

DBMS_OUTPUT.PUT_LINE('kff_filter_purge to delete a valid filter');
l_filterId := 1001;
--valid filter id
fnd_flex_xml_publisher_apis.kff_filter_purge(p_filter_id=>l_filterId);
DBMS_OUTPUT.PUT_LINE('Filter Id: ' || l_filterId);
DBMS_OUTPUT.PUT_LINE('VALID FILTER_ID DELETED SUCCESSFULLY');
This chapter discusses how to configure flexfields with test segments, how to test your flexfield business components in Oracle Fusion applications using Integrated WebLogic Server, how to deploy a flexfield application to a Standalone Oracle WebLogic Server Environment, how to deploy flexfields, how to regenerate flexfield business components programmatically, and how to make flexfield setup task flows accessible from Oracle Fusion Functional Setup Manager.

This chapter includes the following sections:

- Introduction to Testing and Deploying Flexfields
- Deploying and Running the Flexfield Configuration User Interface
- Configuring Flexfields
- Testing Flexfields
- Deploying Flexfields in a Standalone WebLogic Server Environment
- Using the WLST Flexfield Commands to Deploy Flexfields
- Regenerating Flexfield Business Components Programmatically
- Integrating Flexfield Task Flows into
- Exporting Flexfield Business Components

27.1 Introduction to Testing and Deploying Flexfields

After you create a flexfield and verify that the pages on which you added the flexfield run without errors, you will want to test the flexfield. You can test the flexfield in one of three ways:

- Using the Run with Flexfields dialog or the Debug with Flexfields dialog from Oracle JDeveloper as described in Testing Flexfields from Oracle JDeveloper.
- Using an Integrated WebLogic Server instance as described in Testing Flexfields from Integrated WebLogic Server.
- Deploying your application to a standalone Oracle WebLogic Server instance as described in Deploying Flexfields in a Standalone WebLogic Server Environment. With this method, you can do end-to-end testing. That is, you can configure a flexfield, deploy it, and view the pages that contain the flexfield.

If you want to test a flexfield from JDeveloper or an Integrated WebLogic Server instance and the flexfield does not have any segments configured, you must first configure it as described in Configuring Flexfields so that it will appear in its pages.
Note:

If you do not have access to an application that provides the manage flexfield tasks, deploy and run a UI that provides access as described in Deploying and Running the Flexfield Configuration User Interface.

An alternative method for deploying flexfield configurations is to use Oracle WebLogic Scripting Tool (WLST) commands as described in Using the WLST Flexfield Commands to Deploy Flexfields.

In addition to using the Create Flexfield Business Components wizard to generate flexfield business components, you can create a Java program to invoke the flexfield business component modeler and thus automate the creation of the business components as described in Regenerating Flexfield Business Components Programmatically.

Integrating Flexfield Task Flows into describes the available flexfield tasks flows for integration into Oracle Fusion Functional Setup Manager.

27.2 Deploy and Run the Flexfield Configuration User Interface

To test a flexfield, the flexfield must have configured segments so that the flexfield appears in the application user interface (UI). If you need to configure a flexfield for testing from Integrated WebLogic Server, you can deploy the Applications Core (ApplCore) Setup application to an Integrated WebLogic Server instance, and then use one of the manage flexfield tasks in that application to configure the flexfield.

After you access the manage flexfield tasks, configure the flexfield as described in Configuring Flexfields.

27.2.1 How to Deploy and Run the ApplCore Setup Application to Integrated WebLogic Server

Use the fndSetup.ear file to deploy the ApplCore Setup application to an Integrated WebLogic Server instance.

Note:

You can use this deployed application to configure flexfields but you cannot use it to deploy the configurations. However, as you will generate the flexfield business components using the Tester role as described in Testing Flexfields from Integrated WebLogic Server, you do not need to deploy the configurations.

Before you begin:

- Although you will not deploy the configurations to an MDS repository, you must provide the name of a valid MDS repository and JNDI data source in the Applcore Setup application's adf-config.xml file. For information about MDS repositories, see the “Managing the Metadata Repository” chapter in the Administering Oracle Fusion Middleware.

- If you have not started an Integrated WebLogic Server instance from JDeveloper, choose Run and then choose Start Server Instance (Integrated WebLogic Server).
To deploy and run the ApplCore Setup application:

1. Copy `jdev_install/jdeveloper/jdev/oaext/external/FndSetup.ear` to a directory.

2. Complete the following steps to specify the Oracle Metadata Services (MDS) repository partition to use for the flexfield configuration metadata.
   a. Change to the directory that contains the copy of the `FndSetup.ear` file and complete the following commands to expand the enterprise archive (EAR) file.
      
      ```
      mkdir tmpDir
      cd tmpDir/
      unzip ../FndSetup.ear
      ```
   
   b. Open the `tmpDir/adf/META-INF/adf-config.xml` file in an editor and insert the `metadata-store` element shown in bold in Example 27-1 into the `metadata-store-usage` section.
      
      Replace `repository-name` with the name of an existing MDS repository. Replace `jndi-data-source` with a valid JNDI data source. The JNDI data source name is typically based on the repository name. For example the repository name might be `mds-ApplicationMDSDB` and the JNDI data source might be `jdbc/mds/mds-ApplicationMDSDBDS`.
      
      Replace `test-partition-name` with either the name of an existing partition or the name of a partition to create for the test configuration metadata.
   
   c. Change to the `tmpDir` folder.
   
   d. Execute the following commands to re-create the EAR file with the modified `adf-config.xml` file.
      
      If you are using a Linux operating system, use these commands:
      
      ```
      rm ../FndSetup.ear
      zip -r FndSetup.ear .
      mv FndSetup.ear ../
      rm -Rf tmpDir
      ```
      
      If you are using a Windows operating system, use these commands:
      
      ```
      rmdir /S /Q ..\FndSetup.ear
      zip -r FndSetup.ear .
      move FndSetup.ear ..\
      rmdir /S /Q tmpDir
      ```

3. Log in to the administration console for the Integrated WebLogic Server instance. Note that the URL for administration is commonly set to `http://localhost:7101/console`.

4. Ensure that the `FUSION_APPS_FSCM_ASYNC_WS_APPID` user principal exists in the security realm.
   a. From the Domain Structure, click Security Realm and then click `myrealm` in the Realms table.
   
   b. Click the Users and Groups tab and click the Users tab.
   
   c. If you do not see an entry for the name `FUSION_APPS_FSCM_ASYNC_WS_APPID`, click New, type
FUSION_APPS_FSCM_ASYNC_WS_APPID in the Name field, select the DefaultAuthenticator provider, type a description and a password and click OK.

5. From the Domain Structure in the administration console, click Deployments.

6. From the Summary of Deployments, click Install.

7. In the Path field, type the full path to the directory that contains the FndSetup.ear file with the modified adf-config.xml file.

8. Select FndSetup.ear and click Next.

9. Select Install this deployment as an application and click Next.

10. Click Finish.

11. Use the following URL to run the application in a browser. Replace host and port with the appropriate host and port, such as localhost and 7101. Log in with the application_developer user name and the Welcome1 password.

http://host:port/fndSetup/faces/SetupDemo_UIShellPage

See Configuring Flexfields for information about using the application to configure the flexfields.

Example 27-1  metadata-store Element to Add to the adf-config.xml File

```xml
<metadata-store-usages>
  <metadata-store-usage id="WebCenterFileMetadataStore"
    deploy-target="true" default-cust-store="true">
    <metadata-store
      class-name="oracle.mds.persistence.stores.db.DBMetadataStore">
      <property value="repository-name" name="repository-name"/>
      <property value="test-partition-name" name="partition-name"/>
      <property value="jndi-data-source" name="jndi-datasource"/>
    </metadata-store>
  </metadata-store-usage>
</metadata-store-usages>
```

27.3 Configure Flexfields

Before you can test a flexfield, you must configure flexfield segments so that the flexfield appears in the UI.

For information about implementing flexfields, do the following:

- Access the Oracle Fusion Applications Technology library.
- See the Implementing Common Features guides for your product family. Search for flexfields.

27.3.1 How to Configure Flexfields for Testing

Use the ApplCore Setup application to access the manage flexfield tasks to configure a flexfield for testing.

Before you begin:

Deploy and run the ApplCore Setup application as described in Deploying and Running the Flexfield Configuration User Interface.
To configure a flexfield for testing:

For testing purposes, use the manage flexfield tasks to create test segments.

For information about using manage flexfield tasks to create test segments, do the following:

- Access the Oracle Fusion Applications Technology library.
- See the Implementing Common Features guides for your product family. Search for flexfields.

If you are testing the flexfields from JDeveloper as described in Testing Flexfields from or from a WebLogic Integrated Server instance as described in Testing Flexfields from Integrated WebLogic Server, do not deploy the flexfield configuration.

27.4 Test Flexfields

You can test flexfields from JDeveloper or from Integrated WebLogic Server.

27.4.1 Test Flexfields from Oracle JDeveloper

The Run with Flexfields and Debug with Flexfields dialogs enable you to test specific flexfields in an application, unbounded task, or JSF page.

The first time that you run either dialog from a JDeveloper instance, the table is empty. You can add and remove flexfields as desired and you can select which flexfields to include in a run.

When you add a flexfield, its status is set to Not Synced, meaning that you must synchronize it before you can include it in a run. When you synchronize a flexfield, its business components are generated using the definitions from the current database connection. In addition, all entity usages are generated and the synchronized artifacts are stored in a local directory. If a flexfield cannot be synchronized successfully, it shows a status of Error, as shown in Figure 27-1.

When you open the dialog, the Sync Status shows the state at the time the dialog was last closed, as long as the database connection has not changed. If the database connection has changed, then the status of every flexfield is changed to Not Synced. If you changed the flexfield after the value shown for Sync Date, then you should synchronize the flexfield again before using it in a run.

When you click Run or Debug, all flexfields that have been selected to include in the run and that have a status of Success will be included in the run.
Before you begin:

1. Ensure that all required entities, view links, application modules, and so on, exist for the flexfield usages that you are testing.

2. Create test segments for the flexfield as described in Configuring Flexfields. The test segment configurations must exist in the ApplicationDB database for the project.

To test flexfields from JDeveloper:

1. From the Application Navigator, right click the project, unbounded task, JSF page, or file that you want to test (the target) and choose Run with Flexfields or choose Debug with Flexfields.

2. In the Run with Flexfields dialog (or the Debug with Flexfields dialog), click the Add icon to add flexfields to the table.

3. Optionally, select a flexfield and click the Remove icon to remove an unnecessary flexfield from the table.

4. If a flexfield that you want to test does not have a Sync Status of Success or the flexfield configuration was changed after the last time you synchronized it, select the flexfield and click the Synchronize icon to generate flexfield business components based on the current flexfield configuration in the database that is displayed for the Database Connection.

Note:

The Sync Status value for existing flexfield entries in the table reflects the status at the time the dialog was last closed. If you have made changes to a flexfield, you must synchronize it again even though it shows a Sync Status of Success.

5. Ensure that the Include in run check boxes for the flexfields that you want to test are selected.

6. Click Run (or click Debug if you are in the Debug with Flexfields dialog) to run the target in an Integrated WebLogic Server instance.
27.4.2 Test Flexfields from Integrated WebLogic Server

To test a flexfield in an Integrated WebLogic Server instance, you make it available for testing and then you use a metadata archive (MAR) profile to package and deploy the application.

27.4.2.1 How to Make Flexfields Available for Testing

Make a flexfield available for testing in an Integrated WebLogic Server instance by either importing the flexfield business component from an existing library or by generating the flexfield business component using the Tester role.

Before you begin:

1. Ensure that all required entities, view links, application modules, and so on exist either in your project, or in a library that is included in your project. Ensure that the ApplicationDB connection points to the database that contains the metadata for the flexfield that you want to test.

2. Create test segments for the flexfield as described in Configuring Flexfields.

3. If you are importing flexfield components from an existing library, the flexfield business components must be regenerated after you configure the flexfield.

To make flexfields available for testing:

You make a flexfield available for testing by doing one of the following:

- Import the flexfield business components from an existing library.
- Generate the flexfield business components that you want to test.

If you use the Create Flexfield Business Components wizard, select the Tester role on the Role page of the wizard, and specify a location for the generated business components. For more information about using the Create Flexfield Business Components wizard, see the appropriate section for the type of flexfield that want to test:

- How to Create Descriptive Flexfield Business Components
- Creating Extensible Flexfield Business Components
- How to Create Key Flexfield Business Components

27.4.2.2 How to Test Flexfields From Integrated WebLogic Server

To test a flexfield in an Integrated WebLogic Server instance, package, deploy, and run the application using a MAR profile. The MAR profile that you use points to the test business component artifacts, such as view objects, entity objects, page fragments, and task flows.

For more information about using MAR profiles, see the "How to Run an Application with Metadata in Integrated WebLogic Server" section in the Developing Fusion Web Applications with Oracle Application Development Framework.

Before you begin:

Create the test business component artifacts for the flexfield, as described in How to Make Flexfields Available for Testing.
To test a flexfield:

1. From the Application menu, choose Application Properties.

2. In the Application Properties dialog, click the Deployment navigation tab and click New to create a new deployment profile.

3. Select MAR File from the Archive Type dropdown list.

4. Enter a name for the profile, and click OK.

5. In the panel on the left-hand side of the Edit MAR Deployment Profile Properties dialog, under MAR Options, select Metadata File Groups and click New.

6. In the Create File Group dialog, enter a name for the user metadata group, as shown in Figure 27-2, and click OK.

   **Figure 27-2  New File Group in MAR Profile Properties**

7. In the User Metadata Group section, add a contributor, enter the path to your test components (a directory or archive), and click OK.

8. In the panel on the left-hand side of the Edit MAR Deployment Profile Properties dialog, under MAR Options, expand Metadata File Groups, expand the user metadata group that you just added, and select the Directories node.

9. In the Directories section, select the root package directory of the flexfield to be tested, as shown in Figure 27-3, and click OK.
Caution:

Ensure that you select the correct item. You must select the directory of the root package of the flexfield to be tested, such that only the objects that are below the level of that package come from the test directory. The root package should match the package that you previously registered with the business component's usage.

As soon as you select the package, Oracle JDeveloper automatically selects the parents all the way to the top of the folder hierarchy, but that does not mean everything under oracle is registered with Oracle Metadata Services (MDS).

10. In the Application Properties dialog, expand Run and click the MDS navigation tab.

11. Select the MAR profile that you created earlier, and click OK.

12. Test your application with the MAR profile.

Note:

If you want to run the Business Component Browser with the business components that you created for testing, you must create a temporary user library that points to the test components, and include the library in your project as the first library. This is because the flexfield view objects are generated into a temporary directory outside the scope of the application workspace.
27.5 Deploy Flexfields in a Standalone WebLogic Server Environment

You can package and deploy your flexfield Oracle Fusion application to a standalone instance of the Oracle WebLogic Server for performing end-to-end testing.

27.5.1 How to Package a Flexfield Application for Deployment

Just as with other Oracle Fusion applications, you must generate an enterprise archive (EAR) file for deployment to an instance of WebLogic Server. Before generating the EAR file, you must enable the flexfield packaging plugin.

27.5.1.1 Enable the Flexfield Packaging Plugin

The flexfield packaging plugin is required to package flexfields from either JDeveloper or the command line. This plugin maps namespaces to the Oracle Metadata Services (MDS) partition.

You enable the flexfield packaging plugin for your working environment by setting the `FLEX_DEPLOY_ADDIN_ENABLED` environment variable set to `true`. For example, in a C shell environment you would run the following command:

```
setenv FLEX_DEPLOY_ADDIN_ENABLED true
```

27.5.1.2 Generate an EAR File for the Application

To make the flexfield artifacts available at runtime, package them into the application's EAR file, which subsequently can be installed on the target server.

Before you begin:

- Complete ADF Business Components models underlying the flexfields, and ensure that all required entities, view links, application modules, and so on exist either in your project, or in a library that is included in your project.
- Enable the flexfield packaging plugin as described in Enabling the Flexfield Packaging Plugin.

To generate an EAR file:

1. From an environment in which the `FLEX_DEPLOY_ADDIN_ENABLED` environment variable has been set to `true`, complete one of the following steps:
   - From the Application Navigator in JDeveloper, right-click the application and choose **Deploy > deployment profile > to EAR file**. When the generation process is done, you can find the path to the generated EAR file in the deployment log message window.

   **Note:**

   You might need to restart JDeveloper to ensure that the `FLEX_DEPLOY_ADDIN_ENABLED` environment variable has taken effect and that the flexfield packaging plugin is enabled.

   When the plugin is enabled in JDeveloper, you will see log entries with a Flexfield prefix, such as [09:25:01 PM] Flexfield: Search library "Applications Core"...
For more information about generating EAR files from JDeveloper, see the "Deploying Fusion Web Applications" chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

- From the command line, run the following `ojdeploy` command:

```bash
jdev_install\jdeveloper\jdev\bin\ojdeploy -profile deployment-profile \ -forcerewrite -workspace application-jws-path
```

2. Optionally, unzip the EAR file and inspect the `adf-config.xml` file to verify that the flexfield packaging plugin added the flexfield packages to the `sessiondef-config` tag and mapped all the flexfield ADF Business Components packages to a `metadata-store-usages` tag. Example 27-2 shows sample tag entries.

**Example 27-2  adf-config.xml Flexfield Tag Entries**

```xml
<adf-config
xmlns="http://xmlns.oracle.com/adf/config"
xmlns:adf="http://xmlns.oracle.com/adf/config/properties" ...
...
<mdsC:sessiondef-config>
<mdsC:package
value="oracle.apps.fnd.applcore.crmdemo.cma.model.view.link.flex;oracle.apps.fnd.applcore.
flex.test.model.entity.flex"/>
</mdsC:sessiondef-config>
...
<mds:mds-config version="11.1.1.000">
<mds:persistence-config>
<mds:metadata-store-usages>
<mds:metadata-store-usage
default-cust-store="true"
deploy-target="true"
id="WebCenterFileMetadataStore">
</mds:metadata-store-usage>
</mds:metadata-store-usages>
</mds:persistence-config>
</mds:mds-config>
...
</adf-config>
```

### 27.5.2 How to Deploy a Flexfield Application

Deploying a flexfield application to an instance of WebLogic Server requires additional steps to ensure that all the flexfields' customization metadata is stored in the proper MDS partition.

For more information about MDS configurations and deploying applications, see the "Deploying the Application" section in the Developing Fusion Web Applications with Oracle Application Development Framework.

The process for deploying a flexfield application includes the following tasks:

1. Prepare the application for deployment.
   
   a. Create an MDS partition to store flexfield customization metadata.
   
   b. Map the product application's EAR file to the MDS partition.
   
   c. Map the Oracle Fusion Middleware Extensions for Applications (ApplCore) Setup application EAR file to the MDS partition.
Deploy Flexfields in a Standalone WebLogic Server Environment

27.5.2.1 Create an MDS Partition

An MDS repository is used to store the information that enables implementors to customize applications. The flexfield packaging process requires a partition in the MDS repository that is associated with the application. You can create a partition specifically for flexfields or you can use the partition for other purposes as well. However, all Oracle Fusion applications and setup applications must use the same MDS partition for all flexfield artifacts.

If the application’s MDS repository does not have the desired partition, use Oracle WebLogic Scripting Tool (WLST) to create one.

For more information about creating and registering MDS repositories and working with WLST, see the "Managing the Metadata Repository" chapter in the Administering Oracle Fusion Middleware.

Before you begin:

Log in to the administration console for the WebLogic Server instance and verify that a Java Database Connectivity (JDBC) data source exists for the MDS repository. This data source is typically named mds-ApplicationMDSDB. Note that the URL for administration is commonly set to http://localhost:7101/console.

For more information about managing JDBC data sources, see the "Creating and Managing JDBC Data Sources" section in the Administering Oracle Fusion Middleware.

To Create an MDS Partition:

1. At the command line, enter the following line to start WLST.

   ```
   sh jdev_install/oracle_common/common/bin/wlst.sh
   ```

   On Windows, use `wlst.cmd`.

2. Enter the following WLST command to connect to the WebLogic Server instance, replacing the user name and password arguments with your user name and password.

   ```
   connect('wls_username', 'wls_password', 'wls_uri')
   ```

   The values must be wrapped in single-quotes. The `wls_uri` value is typically `t3://localhost:7101`.

3. Enter the following WLST command to create the partition.

   ```
   createMetadataPartition('mds_jdbc_data_source', 'partition_name')
   ```

   The `mds_jdbc_data_source` is the JDBC data source for the MDS repository. The `partition_name` can be any string. You might want to consult with your operations team or release team for suggested partition names.
4. Disconnect from WLST.
   
   ```java
   disconnect()
   ```

27.5.2.2 Map the EAR File to the MDS Partition

To configure the application to store the flexfield customization metadata in the desired MDS partition, use the flexfield packaging plugin to update the application’s `adf-config.xml` file with the partition name.

Before you begin:

- Ensure that the `FLEX_DEPLOY_ADDIN_ENABLED` environment variable is set to `true` in your working environment, as described in Enabling the Flexfield Packaging Plugin.
- Generate the EAR file as described in Generating an EAR File for the Application.
- Ensure that an MDS partition exists for the flexfield metadata. For more information, see Creating an Partition.
- Obtain the Java Naming and Directory Interface (JNDI) name for the MDS data source. To find this value, from the administration console for WebLogic Server instance, choose Services, then choose JDBC, and then choose Data Sources.

To map the EAR file to the MDS partition:

1. At the command line, enter the following line to start WLST.

   ```bash
   sh jdev_install/oracle_common/common/bin/wlst.sh
   ```

   On Windows, use `wlst.cmd`.

2. Enter the following WLST command to connect to the WebLogic Server instance, replacing the user name and password arguments with your user name and password.

   ```java
   connect('wls_username', 'wls_password', 'wls_uri')
   ```

   The values must be wrapped in single-quotes. The `wls_uri` value is typically `t3://localhost:7101`.

3. From WLST, execute the following commands to configure the `metadata-store` element. If the element does not exist, the commands add it. If the element already exists, the commands update it.

   ```java
   archive=getMDSArchiveConfig('product_EAR_file_pathname')
   archive.setAppMetadataRepository('mds_jdbc_data_source', 'partition_name', 'DB', 'mds_datasource_JNDI', None)
   archive.save()
   ```

   The `mds_jdbc_data_source` is the JDBC data source for the MDS repository. The `partition_name` is the name of the MDS partition that you are using to store the flexfield customization metadata for all your Oracle Fusion applications. You might need to ask your operations team or release team for the partition name. The
mds_datasource_JNDI is the JNDI name for the MDS data source, such as
jdbc/mds/mds-ApplicationMDSDBDS. For more information, see the
"registerMetadataDBRepository" section in the WLST Command Reference for
WebLogic Server.

4. Optionally, unzip the EAR file and inspect the adf-config.xml file to verify that
the flexfield packaging plugin updated the metadata-store-usage tags to add
the partition name. Example 27-3 shows sample tag entries.

Example 27-3    adf-config.xml metadata-store-usage tags with Added Partition Name

```xml
<mds:metadata-store-usages>
  <mds:metadata-store-usage default-cust-store="true"
deploy-target="true" id="WebCenterFileMetadataStore">
    <mds:metadata-store class-name="oracle.mds.persistence.stores.db.DBMetadataStore">
      <mds:property value="mds-ApplicationMDSDB" name="repository-name"/>
      <mds:property value="ffpartition" name="partition-name"/>
      <mds:property value="jdbc/mds/mds-ApplicationMDSDBDS" name="jndi-datasource"/>
    </mds:metadata-store>
  </mds:metadata-store-usage>
</mds:metadata-store-usages>
```

27.5.2.3 Map the ApplCore Setup Application to the MDS Partition

To test a flexfield, you must configure the flexfield in the same manner as an
implementor would configure it. You can use the ApplCore Setup application to
complete the flexfield configurations as described in How to Configure Flexfields for
Testing. Before using the ApplCore Setup application, you must configure the
application to store the flexfield customization metadata in the desired MDS partition,
just as you would with the product application’s EAR file.

Before you begin:

- Ensure that the FLEX_DEPLOY_ADDIN_ENABLED environment variable is set to
true in your working environment, as described in Enabling the Flexfield
  Packaging Plugin.

- Ensure that an MDS partition exists for the flexfield metadata. For more
  information, see Creating an Partition.

- Locate the ApplCore Setup application’s FndSetup.ear file. This file can
typically be found in the jdev_install/jdeveloper/jdev/oaext/external
directory.

- Obtain the JNDI name for the MDS data source. You can find this value in the
  Services > JDBC > Data Sources page in the administration console for the
  WebLogic Server instance.

To map the ApplCore Setup EAR file to the MDS partition:

1. At the command line, enter the following line to start WLST, if it is not currently
   running.

   ```bash
   sh jdev_install/oracle_common/common/bin/wlst.sh
   ```

   On Windows, use wlst.cmd.
2. If you have not yet connected to the server, enter the following WLST command to connect to the WebLogic Server instance, replacing the user name and password arguments with your user name and password.

```java
connect('wls_username', 'wls_password', 'wls_uri')
```

The values must be wrapped in single-quotes. The `wls_uri` value is typically `t3://localhost:7101`.

3. From WLST, execute the following commands.

```java
archive=getMDSArchiveConfig('ApplCore_Setup_EAR_file_pathname')
archive.setAppMetadataRepository('mds_jdbc_data_source', 'partition_name', 'DB', 'mds_datasource_JNDI', None)
archive.save()
```

The `mds_jdbc_data_source` is the JDBC data source for the MDS repository. The `partition_name` is the name of the MDS partition that you are using to store the flexfield customization metadata for all your Oracle Fusion applications. You might need to ask your operations team or release team for the partition name. The `mds_datasource_JNDI` is the JNDI name for the MDS data source, such as `jdbc/mds/mds-ApplicationMDSDBDS`.

### 27.5.2.4 Include Product Application Model Libraries in the ApplCore Setup EAR File

Before you deploy the ApplCore Setup EAR file, ensure that it contains all the model libraries that are required for your product application.

To include the product application model libraries in the ApplCore Setup EAR file:

1. In a terminal window, change to the directory that contains the `FndSetup.ear` file. This file can typically be found in the `jdev_install/jdeveloper/jdev/oaext/external` directory.

2. Execute the following commands to expand the EAR file.

   ```sh
   mkdir tmpDir
   cd tmpDir/
   unzip ../FndSetup.ear
   ```

3. If the `APP-INF/lib` folder does not exist, execute the following commands to create it.

   ```sh
   mkdir APP-INF
   mkdir APP-INF/lib
   ```

4. Copy all the library Java archive (JAR) files that your product requires to the `APP-INF/lib` folder.

5. Change to the `tmpDir` folder.

6. Execute the following commands to re-create the EAR file with the added JAR files. If you are using a Linux operating system, run these commands:
27.5.2.5 Deploy the Product and Setup Applications to the Server Domains

After you have mapped the applications as described in Mapping the EAR File to the MDS Partition and Mapping the ApplCore Setup Application to the MDS Partition and you have included the project model libraries in the ApplCore Setup application as described in Including Product Application Model Libraries in the ApplCore Setup EAR File, you can deploy the applications to the appropriate domains for your topology.

Related Links
The following documents provide additional information related to subjects discussed in this section:

- For information about creating domains, see the "Creating a WebLogic Domain" chapter in Creating WebLogic Domains Using the Configuration Wizard.
- For information about installing EAR files, see the "Install an Enterprise Application" section in the Oracle WebLogic Server Administration Console Online Help.

27.5.2.6 Prime the MDS Partition with Configured Flexfield Artifacts

The flexfield application is configured to obtain the flexfield customization metadata from the MDS partition. However, no one can log in to the application until the application has gone through an initial process to translate the flexfield metadata into artifacts that are stored in the partition. This task must be completed, even if there is no customization metadata yet. You use WLST to perform this task.

Before you begin:

Deploy the product and setup applications, as described in Deploying the Product and Setup Applications to the Server Domains.

To prime the MDS partition with flexfield metadata artifacts:

- Run the deployFlexForApp WLST command as described in How to Prepare Your Environment to Use the deployFlexForApp Command.

27.6 Use the WLST Flexfield Commands to Deploy Flexfields

You can use the Manage Key Flexfields, Manage Descriptive Flexfields, and Manage Extensible Flexfields tasks to deploy flexfields. In addition, the following WLST commands are available for priming the MDS repository with seeded flexfield artifacts and for deploying flexfields:
• **deployFlexForApp**: Use this command to prime the MDS repository with seeded flexfield artifacts. Deploys all flexfields that do not have a status of DEPLOYED. You can also use this command to deploy all flexfields regardless of their status by setting the `force` parameter to `'true'`.

• **deployFlex**: Use this command to deploy a single flexfield. Deploys the flexfield regardless of status.

• **deleteFlexPatchingLabels**: Use this command to inquire about or delete all flexfield patching labels.

• **validateFlexDeploymentStatus**: Use this command to verify that all flexfields have been successfully deployed. It creates a list of the flexfields that are ready to deploy or failed to deploy.

For information about using WLST flexfield commands, do the following:

• Access the *Oracle Fusion Applications Technology* library.

• See the Implementing Common Features guides for your product family. Search for flexfields.

For information about using the WLST command-line scripting interface, see the *Oracle Fusion Middleware Understanding the WebLogic Scripting Tool* guide.

Before you can use any WLST flexfield command from a development environment, you must first prepare the environment as described in How to Prepare Your Environment to Use the WLST Flexfield Commands. If you use the `deployFlexForApp` command in a development environment, you must complete additional steps described in How to Prepare Your Environment to Use the `deployFlexForApp` Command.

After you execute a WLST flexfield command, you can log into the application and view the pages that contain flexfields. If you have seeded any flexfield configurations by defining value sets, segments, contexts, or structures, for example, the flexfields should appear on the appropriate pages. If a flexfield has not been configured, the corresponding user interface sections will be blank.

### 27.6.1 How to Prepare Your Environment to Use the WLST Flexfield Commands

Before you can use the WLST flexfield commands, you must prepare your environment. The commands will not work until these steps are completed.

**To prepare your environment for WLST flexfield commands:**

1. The WLST flexfield commands can be executed only on the Administration Server for a domain that has a running instance of the ApplCore Setup application. For information on deploying the ApplCore Setup application, see Deploying the Product and Setup Applications to the Server Domains.

2. Ensure that the AppMasterDB data source is registered as a JDBC data source with the Administration Server and points to the same database as the ApplicationDB data source.

### 27.6.2 How to Prepare Your Environment to Use the `deployFlexForApp` Command

The `deployFlexForApp` command translates the product application’s seeded flexfield metadata into artifacts in the MDS repository. This command must be run after you configure your application to read the flexfield artifacts from the MDS.
repository and before you log into the application for the first time, even if there is no seeded flexfield metadata. For more information, see Priming the Partition with Configured Flexfield Artifacts.

To prepare your environment to use the deployFlexForApp command:

1. Configure the product application to store the flexfield customization metadata in the desired MDS partition as described in Mapping the EAR File to the MDS Partition.

2. Map the setup application as described in Mapping the ApplCore Setup Application to the MDS Partition.

3. Deploy the product application and the ApplCore Setup application as described in Deploying the Product and Setup Applications to the Server Domains.

4. Prepare your environment as described in How to Prepare Your Environment to Use the WLST Flexfield Commands.

27.7 Regenerate Flexfield Business Components Programmatically

After you complete the flexfield development activities to incorporate a flexfield into your application, you might need to update the flexfield implementation in your application at a later time by repeating the process of creating the flexfield business components.

To re-create the flexfield business components, you can use the Create Flexfield Business Components wizard, which invokes the flexfield business component modeler to generate (or regenerate) the business components. Alternatively, you can create a Java program to invoke the flexfield business component modeler and thus automate the creation of the business components without implementor interaction.

The flexfield business component modeler can be invoked through the Java application programming interface (API) only in a deployed web application. You must have the following artifacts:

- The ADF Business Components objects required for generating the flexfield business components. Typically these are entity objects. You can deploy them in a JAR file.
- The MDS repository for the generated flexfield business components. If you use a file-system based repository, the metadata path must be an existing writable path.

You either can create a new application to invoke the modeler, or, if you already have a web application for testing, you can include the Java code required for invoking the modeler in your application. The deployment process is the same as any other web application.

Your project must be set up correctly for the program to run successfully. The project configuration requirements are the same as that for the Create Flexfield Business Components wizard.

Example 27-4 demonstrates appropriate Java code for updating the business components for a descriptive flexfield.

Examine BCMModeler.Option for usage-specific settings.
Example 27-4  Java Code for Invoking the Flexfield Business Component Modeler

```java
import oracle.apps.fnd.appcore.flex.runtime.util.BCModeler;

public class Example {
    public static void main(String[] margs) {
        BCModeler.Arguments modelerArgs = new BCModeler.Arguments();

        modelerArgs.put(BCModeler.Parameter.CONNECTION_URL,
                        "jdbc:oracle:thin:user/pass@myhost.com:1999:dev1");

        // Specify where the objects should be.
        modelerArgs.put(BCModeler.Option.OUTPUT_PATH,
                         "/mytest/mds/");

        // The owner of the flexfield should have the following information.
        modelerArgs.put(BCModeler.Parameter.FLEXFIELD_TYPE, "DFF");
        modelerArgs.put(BCModeler.Parameter.APP_SHORT_NAME, "FND");
        modelerArgs.put(BCModeler.Parameter.FLEXFIELD_CODE, "FLEX_DFF1");
        modelerArgs.put(BCModeler.Parameter.TABLE_USAGE_CODE, "FLEX_DFF1");
        modelerArgs.put(BCModeler.Parameter.TABLE_NAME, "FND_DF_TEST_DFF1_T1");
        modelerArgs.put(BCModeler.Parameter.ENTITY_DEF_FULL_NAME,
                        "oracle.apps.fnd.appcore.flex.test.model.entity.Dff1EO");

        // See BCModeler.Option for more usage-specific options. Consult the owner
        // of the flexfield to see if any usage-specific option is required.
        Exception e = BCModeler.run(modelerArgs.getCommandLineArgs(false),
                                     System.out);
        if (e != null) {
            e.printStackTrace();
        }
    }
}
```

27.8 Integrate Flexfield Task Flows into Oracle Fusion Functional Setup Manager

Every Oracle Fusion application registers task flows for setup activities with a product
called Oracle Fusion Functional Setup Manager. For example, a human resources (HR)
application can register setup activities such as "Create Employees" and "Manage
Employee Tree Structure." Implementors and administrators use these registered task
flows, which are accessed from the Oracle Fusion Applications Setup and Maintenance
work area of Oracle Fusion Applications, to configure the applications by defining
custom configuration templates or tasks based on their business needs.

Note:
The registration application task flow is not available for extensible flexfields
and key flexfields. You must use the FND_FLEX_DF_SETUP_APIS PL/SQL to
register extensible flexfields as described in Defining and Registering
Extensible Flexfields. You must use the FND_FLEX_KF_SETUP_APIS
PL/SQL API to register key flexfields as described in How to Develop Key
Flexfields.
Table 27-1 lists the flexfield setup task flows. To make these task flows available to developers, implementors, or administrators, register the appropriate task.

For information about implementing your specific product family, do the following:

- Access the Oracle Fusion Applications Technology library.
- See the Implementing Common Features guides for your product family. Search for "Define Flexfields".

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Parameters Passed</th>
<th>Behavior</th>
</tr>
</thead>
</table>
| Manage Descriptive Flexfields | /WEB-INF/oracle/apps/fnd/applcore/flex/dff/ui/publicFlow/ManageDescriptiveFlexfieldsTF.xml#ManageDescriptiveFlexfieldsTF | To invoke search mode:  
mode='search'

To restrict search mode to descriptive flexfields belonging to a particular product module:

mode='search'  
moduleType='moduletype'  
moduleKey='modulekey'

To invoke edit mode for a specific descriptive flexfield:

mode='edit'  
descriptiveFlexfieldCode=dffcode  
applicationId=appid

To optionally specify a page heading for the task flow:

pageTitle='titlestring' | Search and edit descriptive flexfield configuration. |
### Table 27-1 (Cont.) Flexfields Task Flows and Parameters

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Parameters Passed</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Extensible Flexfields</td>
<td>/WEB-INF/oracle/apps/fnd/aplicore/flex/dff/ui/publicFlow/ManageExtensibleFlexfieldsTF.xml#ManageExtensibleFlexfieldsTF</td>
<td>To invoke search mode:</td>
<td>Search and edit extensible flexfield configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode='search'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>To restrict search mode to extensible flexfields belonging to a particular product module:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode='search' moduleType='moduletype' moduleKey='modulekey'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>To invoke edit mode for a specific extensible flexfield:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode='edit' extensibleFlexfieldCode=effcode applicationId=appid</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>To optionally specify a page heading for the task flow:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>pageTitle='titlestring'</td>
<td></td>
</tr>
<tr>
<td>Manage Key Flexfields</td>
<td>/WEB-INF/oracle/apps/fnd/aplicore/flex/kff/ui/publicFlow/ManageKeyFlexfieldsTF.xml#ManageKeyFlexfieldsTF</td>
<td>To invoke search mode:</td>
<td>Search and edit key flexfield configuration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode='search'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>To restrict search mode to key flexfields belonging to a particular product module:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode='search' moduleType='moduletype' moduleKey='modulekey'</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>To invoke edit mode for a specific key flexfield:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode='edit' keyFlexfieldCode=kffcode applicationId=appid</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>To optionally specify a page heading for the task flow:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>pageTitle='titlestring'</td>
<td></td>
</tr>
</tbody>
</table>
## Table 27-1  (Cont.) Flexfields Task Flows and Parameters

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Parameters Passed</th>
<th>Behavior</th>
</tr>
</thead>
</table>
| Manage Value Sets | /WEB-INF/oracle/apps/fnd/applcore/flex/vst/ui/publicFlow/ManageValueSetsTF.xml#ManageValueSetsTF | To invoke search mode for any value set:  
  mode='search'  

To restrict search mode to value sets belonging to a particular product module:  
  mode='search'  
  moduleType=moduletype  
  moduleKey=modulekey  

To invoke edit mode for a specific value set:  
  mode='edit'  
  valueSetCode=vscode  

To optionally specify a page heading for the task flow:  
  pageTitle='titlestring' | Search and edit flexfield value sets. |

For related information about functional security actions and roles based on task flows, see Implementing Function Security

### 27.9 Export Flexfield Business Components

As explained in Deploying Flexfields in a Standalone WebLogic Server Environment, you can package and deploy your flexfield Oracle Fusion application to a standalone instance of Oracle WebLogic Server to perform end-to-end testing.

Deploying a flexfield generates Business Components and other metadata files in your MDS repository. These flexfield metadata files can be exported from the Metadata repository using Flexfield Task Flows explained in Integrating Flexfield Task Flows into Oracle Fusion Functional Setup Manager.

A menu Actions item, Download Flexfield Archive, is available as part of Manage Descriptive Flexfields, Manage Extensible Flexfields and Manage Key Flexfields taskflows.
Figure 27-4  Download the Flexfield Archive

When Download Flexfield Archive is selected, a dialog with a progress bar is opened where the progress of exporting flexfield metadata files from the Metadata Repository is displayed.

Figure 27-5  Download Progress Display

When Download Flexfield Archive is selected, a dialog with a progress bar is opened where the progress of exporting flexfield metadata files from the Metadata Repository is displayed.
When exporting the flexfield metadata files is completed, an archive is prepared with these artifacts.

The **Download** button becomes enabled on the progress popup. Click **Download** to download the archive file containing the flexfield metadata to the machine on which the browser is running.
This chapter provides an introduction to Oracle Enterprise Crawl and Search Framework (ECSF). It also describes how to set up ECSF.

This chapter includes the following sections:

- **Introduction to Using Oracle Enterprise Crawl and Search Framework**
- **Setting Up and Running ECSF Command Line Administration Utility**
- **Setting Up Oracle Enterprise Manager and Discovering ECSF**

### 28.1 Introduction to Using Oracle Enterprise Crawl and Search Framework

Oracle Enterprise Crawl and Search Framework (ECSF) is an Oracle Fusion Middleware search framework that enables you to quickly expose application context information on various business objects to enable full-text transactional search.

For more information, see the "Managing Search with Oracle Enterprise Crawl and Search Framework" chapter in the *Administrator’s Guide*.

#### 28.1.1 Understanding the ECSF Architecture

ECSF abstracts an underlying search engine and provides a common set of application programming interfaces (APIs) for developing search functionalities. ECSF serves as an integration layer between the search engine and the Oracle Fusion applications. **Figure 28-1** illustrates the architecture of the ECSF framework.
ECSF includes the following high-level components:

- Searchable Object Manager
- Search Designer
- Semantic Engine
- Fusion Applications Control
- ECSF Command Line Administration Utility
- Security Services
- Data Services
- Query Service

ECSF integrates with the Oracle Secure Enterprise Search (Oracle SES) engine to support application search. Oracle SES provides capabilities for crawling and indexing the metadata and objects exposed by ECSF. The Security Plug-in and Crawler Plug-in are modules on Oracle SES that interface with ECSF.

### 28.1.1.1 About Searchable Object Manager

Searchable Object Manager, serving as a metadata manager, manages searchable objects and provides the runtime interface for accessing these objects. At runtime, the Searchable Object Manager loads the searchable objects from persistent storage, validates the searchable object definitions, and provides the searchable objects to the Crawlable Factory component of the Data Service.

The Searchable Object Manager is also responsible for the life cycle management of searchable objects, which administrators can deploy, customize, and enable or disable via the Fusion Applications Control or the ECSF Command Line Administration Utility.

### 28.1.1.2 About Search Designer

The Search Designer is a page in Oracle JDeveloper 11g that provides the interface for defining the metadata that describes the business objects to be indexed. You can also use this design interface to specify the security mechanism used to protect the data, as
well as define the searchable object search characteristics, which include Advanced Search, Faceted Navigation, and Actionable Results.

28.1.1.3 About Semantic Engine

The Semantic Engine leverages the semantic information of searchable object definitions to create context around the search. It achieves this by interpreting the searchable object definitions with relation to the runtime user information during both crawl and query time. Runtime user information may include the following:

- Facets
- Actionable results
- Security
- Personalization
- Internationalization
- Data structure mapping
- Tagging, commenting, rates
- Results clustering
- Context filtering
- Custom weighting

28.1.1.4 About Fusion Applications Control

The Fusion Applications Control is an Oracle Enterprise Manager extension that provides a user interface for registering searchable objects in the ECSF schema in the Oracle Fusion Applications database, and for administering the runtime parameters of ECSF, the target search engine, and the configuration of parameters.

28.1.1.5 About ECSF Command Line Administration Utility

The ECSF Command Line Administration Utility is a standalone command line interface that provides a user interface for registering searchable objects in the ECSF schema in the Oracle Fusion Applications database. You can also use this tool for configuring and administering ECSF without external dependencies on Oracle Enterprise Manager.

28.1.1.6 About Security Service

The Security Service is the runtime server component responsible for providing security information to SES. During query time, this service retrieves the security keys of the user performing the search and passes them to Oracle SES, where they are used to filter the query results.

The Security Service server component is also invoked during crawl time to add security information (access control lists) to data before inserting or creating indexes on the search engine (Oracle SES). An access control list (ACL) is a list that identifies the users who can access the associated object and that specifies the user's access rights to that object. The ACL values generated by the Security Service during crawl time should match the corresponding keys generated during query time.
Note:

In ECSF, the generic term ACL (access control list) is used to describe how Oracle SES and ECSF pass security information and perform security checks by using the information described in the ACL.

The Security Service component is implemented as a security engine with a plug-in interface. The security plug-in determines the format of the ACL keys. For all custom security models, a new Security Plug-in must be implemented. Security Service uses Oracle Platform Security for Java to authenticate users and call the Security Plug-in to retrieve security values for a given searchable object.

For more information about security for ECSF, see Configuring ECSF Security.

28.1.1.7 About Data Service

Data Service is the primary data interface, based on a proprietary Really Simple Syndication (RSS) feed format, between ECSF and the search engine. In addition to supporting the flow of metadata between ECSF and the search engine, Data Service supports attachments, batching, and error handling.

Data Service authenticates each Oracle SES crawl request by using Oracle Platform Security for Java to validate the user credentials and permissions for crawling the data source.

The Crawlable Factory component, part of Data Service, determines how searchable objects are broken down and manages the construction of RSS feeds to the search engine.

28.1.1.8 About Query Service

The Query Service provides a search interface for the applications UI and handles all search requests. This service performs query rewrite, parameter substitution, and other preprocessing operations before invoking the underlying configured search engine.

Search results are also serviced via this service. Hooks are provided to preprocess and postprocess data, which facilitates the capability to filter search results.

28.1.1.9 About the Oracle SES Search Engine

Oracle SES enables a secure, uniform search across multiple enterprise repositories. ECSF integrates with Oracle SES technology to provide full-text search functionality in Oracle Fusion Applications.

Note:

The application server space is demarcated to identify that ECSF runs in a separate application server, outside the search engine. It is recommended, for performance reasons, that each search engine instance runs on separate hardware.

28.1.1.10 About Security Plug-in

Oracle SES provides an API for writing security plug-ins (or connectors) in Java. With this API, you can create a security plug-in to meet your requirements. ECSF Security Service interfaces with this security plug-in. The Security Plug-in invokes the Security
Service to retrieve keys, to which the user has access, for filtering the results that are delivered to the ECSF query service. A proxy user must be set up on the search engine to invoke the Security Service. The proxy user must have security privileges for the Oracle Fusion applications. For more information about security for ECSF, see Configuring ECSF Security.

28.1.1.11 About the Crawler Plug-in

The Crawler Plug-in is a search engine (Oracle SES) module that implements the modified RSS feed format between ECSF and Oracle SES. This component deserializes the data sent from ECSF, via the Data Service component, and interfaces with the Oracle SES components that creates the indexes.

28.2 Setting Up and Running ECSF Command Line Administration Utility

You can use the ECSF Command Line Administration Utility to quickly test and manage the searchable objects without having to use Oracle Enterprise Manager Fusion Applications Control.

Note:

Administrators should use Fusion Applications Control to manage the life cycle of searchable objects in the production environment.

28.2.1 Set Up the ECSF Command Line Administration Utility

Before you can run the utility, you must complete the following setup requirements:

1. Make the searchable objects accessible to the ECSF Command Line Admin Utility.

2. Set the class path to make sure it contains the required Oracle classes. ECSF provides a set of scripts that you can use to set the class path.

3. Connect to the database by performing one of the following tasks:
   • Provide the connection information in the script so that the ECSF Command Line Administration Utility automatically connects to the specified database during startup.
   • Manually connect to the database after you start the ECSF Command Line Administration Utility.

4. Provide the path of the JPS Config file.

5. Create ECSF query proxy users. To perform commands that connect to the Oracle SES server (for example, deploy, start schedule, etc.), the engine instance must be set up correctly so that its parameters have the required information. For more information, see the "Managing Search with Oracle Enterprise Crawl and Search Framework" chapter in the Administrator’s Guide.

6. (Optional) Configure the log settings.

7. (Optional) Set the startup parameter to support taking input from a text file.
28.2.2 Run the ECSF Command Line Administration Utility

After you have set up the ECSF Command Line Administration Utility, you can run the utility by any of the following ways:

- Execute the `runCmdLineAdmin.bat` script (Windows)
- Execute the `runCmdLineAdmin.sh` script (Linux)
- Start it as a Java program from a command line interface with the following command:
  ```
  java oracle.ecsf.cmdlineadmin.CmdLineAdmin
  ```

Enter a username and password when prompted.

---

**Note:**

If you enter an invalid username and password, you can either reconnect manually by using the `connect` command, or exit the ECSF Command Line Administration Utility (type `exit` or press `Ctrl-C`) and try again.

When you update passwords in the Lightweight Directory Access Protocol (LDAP) credential store from the ECSF Command Line Administration Utility, the `jps-config-jse.xml` file must contain the same LDAP information as the `jps-config.xml` file. Java Platform Security (JPS) does not propagate the changes from `jps-config.xml` to `jps-config-jse.xml` automatically.

---

All commands, responses, and error messages in the ECSF Command Line Administration Utility are logged.

To exit the ECSF Command Line Administration Utility, enter the `exit` command at the prompt.

28.2.3 About Making Searchable Objects Accessible to the ECSF Command Line Administration Utility

Make the searchable objects accessible to the ECSF Command Line Administration Utility by adding the ADF library JAR file containing the view object and entity object definitions to its class path.

The ECSF Command Line Administration Utility needs the path of the JAR file containing the searchable objects. These metadata objects are validated during register and unregister operations.

You can find the unpacked EAR files containing the searchable object JAR files for the search applications in the following locations:

- **For Linux OS:** `/net/mount1/appbase/fusionapps/applications/fscm/deploy/EarFscmSearch.ear/app-INF/lib/searchable_object_jar_file`
- **For Windows OS:** `C:\appbase\fusionapps\applications\fscm\deploy\EarFscmSearch.ear\APP-INF\lib\searchable_object_jar_file`
• For Linux OS: /net/mount1/appbase/fusionapps/applications/crm/deploy/EarCrmSearch.ear/APP-INF/lib/searchable_object_jar_file
  For Windows OS: C:\appbase\fusionapps\applications\crm\deploy\EarCrmSearch.ear\APP-INF\lib\searchable_object_jar_file

• For Linux OS: /net/mount1/appbase/fusionapps/applications/hcm/deploy/EarHcmSearch.ear/APP-INF/lib/searchable_object_jar_file
  For Windows OS: C:\appbase\fusionapps\applications\hcm\deploy\EarHcmSearch.ear\APP-INF\lib\searchable_object_jar_file

To add the ADF library JAR file containing the view object and entity object definitions to the class path for the ECSF Command Line Administration Utility, you must first create the ADF library.

The application JAR file, which contains the searchable objects that are defined in your application, is written to the deploy directory of the project.

To deploy or undeploy a searchable object, a JAR file containing the searchable object must be specified in the class path of the ECSF Command Line Administration Utility. For information, see About Setting the Class Path.

If you are deploying searchable objects from multiple applications, you must create a JAR file for each of those applications to add the searchable objects to the class path.

28.2.4 About Setting the Class Path

In order for the ECSF Command Line Administration Utility to run, the class path must contain the required Oracle classes. Modify and run scripts to set the class path, as well as optional connection information, and run the ECSF Command Line Administration Utility.

Note:

If you receive a java.lang.ClassNotFoundException exception, then add the JAR file containing that class to ADMIN_CP in the script.

The ECSF Command Line Administration Utility references the class path to obtain the location of Oracle Library home, Java home, Oracle WebLogic Server home, and the JAR files needed for the ECSF Command Line Administration Utility operations.

28.2.4.1 Set the Class Path in Windows

Modify and run the runCmdLineAdmin.bat script to set the class path in a Windows environment.

To set the class path in Windows, do the following:

1. In a text editor, open the runCmdLineAdmin.bat script from MW_HOME/Oracle_atgpf1/ecsf/modules/oracle.ecsf_11.1.1/admin.

2. Specify the Oracle Library home directory path by locating the line set ORACLE_LIBRARY_HOME=SET_ORACLE_LIBRARY_HOME and replace SET_ORACLE_LIBRARY_HOME with the ATGPF shiphome directory, for example, set ORACLE_LIBRARY_HOME=C:\mw_home\oracle_common.
3. Specify the ATGPF Library home directory path by locating the line `set ATGPF_LIBRARY_HOME=SET_ATGPF_LIBRARY_HOME` and replace `SET_ATGPF_LIBRARY_HOME` with the ATGPF shiphome directory, for example, `set ATGPF_LIBRARY_HOME=C:\mw_home\Oracle_atgpf1`.

Specify the ATGPF Library home directory path by locating the line `set ATGPF_LIBRARY_HOME=SET_ATGPF_LIBRARY_HOME` and replace `SET_ATGPF_LIBRARY_HOME` with the ATGPF Library directory, for example, `set ATGPF_LIBRARY_HOME=c:\fmwtools_view\fmwtools\mw_home\jdeveloper`.

4. Specify the Oracle WebLogic Server home directory path:
   a. Locate the following line: `set WLS_HOME=SET_WLS_HOME`
   b. Replace `SET_WLS_HOME` with the Oracle WebLogic Server home directory path, for example, `set WLS_HOME=C:\MW_HOME/wlserver_10.3`

5. Specify the Java home directory path:
   • Locate the following line: `set JAVA_HOME=SET_JAVA_HOME`
   • Replace `SET_JAVA_HOME` with the Java home directory path (where the Java executable should be located), for example, `set JAVA_HOME=C:\Java\jdk\bin`.

The version of Java used must match the version required by the Oracle build.

6. Specify the directory path of the application JAR file:
   • Locate the following line: `set APP_JAR=SET_APP_JAR`
   • Replace `SET_APP_JAR` with the directory path of the application JAR file you created in About Making Searchable Objects Accessible to the ECSF Command Line Administration Utility, for example, `set APP_JAR=C:\Jdeveloper\mywork\Application1\runtime\deploy\archive1.jar`.

7. Save the script file.

### 28.2.4.2 Set the Class Path in Linux

Modify and run the `runCmdLineAdmin.sh` script to set the class path in a Linux environment.

To set the class path in Linux, do the following:

1. In a text editor, open the `runCmdLineAdmin.sh` script from `MW_HOME/Oracle_atgpf1/ecsf/modules/oracle.ecsf_11.1.1/admin`.

2. Specify the Oracle Library home directory path by locating the line `export ORACLE_LIBRARY_HOME=SET_ORACLE_LIBRARY_HOME` and replace `SET_ORACLE_LIBRARY_HOME` with the ATGPF shiphome directory, for example, `export ORACLE_LIBRARY_HOME="/scratch/mw_home/Oracle_atgpf1"`.

Specify the Oracle Library home directory path by locating the line `export ORACLE_LIBRARY_HOME=SET_ORACLE_LIBRARY_HOME` and replace `SET_ORACLE_LIBRARY_HOME` with the Oracle Common Library directory, for example, `export ORACLE_LIBRARY_HOME="/scratch/login/`
3. Specify the ATGPF Library home directory path by locating the line `export ATGPF_LIBRARY_HOME=SET_ATGPF_LIBRARY_HOME` and replace `SET_ATGPF_LIBRARY_HOME` with the ATGPF shiphome directory, for example, `set ATGPF_LIBRARY_HOME="/scratch/fmwtools/mw_home/Oracle_atgpf1"`.

4. Specify the Java home directory path:
   - Locate the following line: `export JAVA_HOME="set_java_home"
   - Replace `set_java_home` with the Java home directory path (where the Java executable should be located), for example, `export JAVA_HOME="/Java/jdk/bin"`.
   
   The version of Java used must match the version required by the Oracle build.

5. Specify the directory path of the application JAR file:
   - Locate the following line: `export APP_JAR="set_app_jar"
   - Replace `set_app_jar` with the directory path of the application JAR file you created in About Making Searchable Objects Accessible to the ECSF Command Line Administration Utility, for example, `export APP_JAR="/Jdeveloper/mywork/Application1/runtime/deploy/archive1.jar"`.

6. Save the script file.

7. Run the script.

### 28.2.5 About Setting the Connection Information

The ECSF Command Line Administration Utility requires an Oracle Fusion Applications database, to which it can either be directly connected or connected through a remote MBean. To use the ECSF Command Line Administration Utility, you must supply the connection information.

Set the connection information in the `runCmdLineAdmin` script so that the ECSF Command Line Administration Utility automatically connects to the specified database or MBean server during startup. If you do not include the connection information in the script, then you must manually create the connection to the Oracle Fusion Applications database after you start the ECSF Command Line Administration Utility. For information, see About Manually Connecting to the Database.

The information for connecting to the database or MBean server is saved in the `runCmdLineAdmin` script for the ECSF Command Line Administration Utility to use for connecting to the Oracle Fusion Applications database at startup. You are prompted to enter a password after you start the ECSF Command Line Administration Utility.

#### 28.2.5.1 Set the Connection Information in Windows

Modify the `runCmdLineAdmin.bat` script to set the connection information in a Windows environment.

To set the connection information in Windows, do the following:
1. Open the runCmdLineAdmin.bat script, located in JDEV_INSTALL/ecsf, in a text editor.

2. Locate set CONNECT_INFO= and specify the database or MBean server, using one of the following formats:
   - connect to mbeanserver hostname port
   - Using SID:
     connect to database hostname port SID
     For example,
     set CONNECT_INFO=connect to database fusionhost123 1566 fh123.
   - Using service name:
     connect to database service hostname port servicename
     For example,
     set CONNECT_INFO=connect to database service fusionhost123 5521 myservice
   - Using database descriptor:
     connect to database descriptor 'descriptor'
     The descriptor argument must be enclosed in quotation marks and can contain either the SID or service name. For example:
     - Using SID:
       set CONNECT_INFO=connect to database descriptor
       '(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)
       (HOST=fusionhost123)(PORT=5521))
       (CONNECT_DATA=(SID=dbmsdb2)))'
     - Using service name:
       set CONNECT_INFO=connect to database descriptor
       '(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)
       (HOST=fusionhost123)(PORT=5521))(CONNECT_DATA=(SERVICE
       NAME=myservice)))'

3. Save the script file.

28.2.5.2 Set the Connection Information in Linux

Modify and run the runCmdLineAdmin.sh script to set the connection information in a Linux environment.

To set the connection information in Linux, do the following:

1. Open the runCmdLineAdmin.sh script, located in ORACLE_HOME/jdeveloper/ecsf, in a text editor.

2. Locate export CONNECT_INFO= and specify the database or MBean server, using one of the following formats:
   - connect to mbeanserver hostname port
• Using SID:
  connect to database *hostname* port *SID*
  For example,
  set CONNECT_INFO=connect to database fusionhost123 1566 fh123

• Using service name:
  connect to database service *hostname* port *servicename*
  For example,
  set CONNECT_INFO=connect to database service fusionhost123 5521 myservice

• Using database descriptor:
  connect to database descriptor 'descriptor'
  The *descriptor* argument must be enclosed in quotation marks and can contain either the SID or service name. For example:
  - Using SID:
    set CONNECT_INFO=connect to database descriptor
    'DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=fusionhost123)(PORT=5521))
    (CONNECT_DATA=(SID=dmssdb2))'
  - Using service name:
    set CONNECT_INFO=connect to database descriptor
    'DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=fusionhost123)(PORT=5521))
    (CONNECT_DATA=(SERVICE_NAME=myservice))'

3. Save the script file.

### 28.2.6 About Manually Connecting to the Oracle Fusion Applications Database

The ECSF Command Line Administration Utility requires an Oracle Fusion Applications database, to which it can either be directly connected or connected through a remote MBean, for command execution. To use the ECSF Command Line Administration Utility, you must supply the connection information.

You can supply connection information either before or after starting the ECSF Command Line Administration Utility. Supplying the connection information before startup allows the ECSF Command Line Administration Utility to automatically connect to the specified database or MBean server during startup. For information, see About Setting the Connection Information.

If you choose not to supply the connection information before startup, you must manually create the connection to the Oracle Fusion Applications database after you start the ECSF Command Line Administration Utility.

To create a connection to the Oracle Fusion Applications database directly, enter one of the following commands at the ECSF Command Line Administration prompt, then press Enter:

- connect to database
  The ECSF Command Line Administration Utility prompts you for the host name, port, and SID.
- connect to database service
The ECSF Command Line Administration Utility prompts you for the host name, port, and service name.

- **connect to database descriptor**

  The ECSF Command Line Administration Utility prompts you for the descriptor. The `descriptor` argument must be enclosed in quotation marks and can contain either the SID or service name. For example:

  - Using SID:
    
    `'(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=fusionhost123)(PORT=5521))
    (CONNECT_DATA=(SID=dbmsdb2)))'`

  - Using service name:
    
    `'(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=fusionhost123)(PORT=5521))
    (CONNECT_DATA=(SERVICE_NAME=myservice)))'`

  - **connect to database hostname port SID**

    You can directly pass the required values as arguments into the command.

  - **connect to database service hostname port servicename**

    You can directly pass the required values as arguments into the command.

  - **connect to database descriptor 'descriptor'*

    You can directly pass the required value as an argument into the command. The `descriptor` argument must be enclosed in quotation marks and can contain either the SID or service name. For example:

    - Using SID:
      
      `'(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=fusionhost123)(PORT=5521))
      (CONNECT_DATA=(SID=dbmsdb2)))'`

    - Using service name:
      
      `'(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=fusionhost123)(PORT=5521))
      (CONNECT_DATA=(SERVICE_NAME=myservice)))'`

To create a connection to the Oracle Fusion Applications database through a remote MBean, enter one of the following commands at the ECSF Command Line Administration prompt, then press Enter:

- **connect to mbeanserver**

  The ECSF Command Line Administration Utility prompts you for the required values.

- **connect to mbeanserver hostname port**

  You can directly pass the required values as arguments into the command.

The ECSF Command Line Administration Utility prompts you to enter a username and password.

### 28.2.7 How to Provide the Path of the JPS Config File

The JPS Config file (`jps-config-jse.xml`) contains the credential store information needed for running the ECSF Command Line Administration scripts. You must
provide the path of the JPS Config file by modifying the ECSF Command Line Administration scripts.

To set the JPS Config file path, do the following:

1. Open the runCmdLineAdmin.bat (Windows) or runCmdLineAdmin.sh (Linux) script, located in ORACLE_HOME/jdeveloper/ecsf, in a text editor.

2. Set the JPS_CONFIG parameter to point to the location of jps-config-jse.xml (usually at DOMAIN_HOME/config/fmwconfig/jps-config-jse.xml).

3. Save.

### 28.2.8 About Configuring the Log Settings

The scripts for the ECSF Command Line Administration Utility point to the logging configuration file (ecsfcla-logging.xml), where you can configure log settings, such as log level and log file location. The `ecsfcla-logging.xml` file is located in the same directory as the ECSF Command Line Administration scripts (`JDEV_INSTALL/ecsf`). To configure log settings, modify the property values in `ecsfcla-logging.xml` and save the file.

The location of `ecsfcla-logging.xml` can be changed by modifying the `ODL_CONFIG` parameter in the ECSF Command Line Administration scripts.

All commands, responses, and error messages in the ECSF Command Line Administration Utility are logged. The generated log files follow the format `ecsfCmdLineAdminLog*.txt`, and its default location is `JDEV_INSTALL/ecsf/log`.

### 28.2.9 About Automating the ECSF Command Line Administration Utility

You can set the ECSF Command Line Administration Utility to automatically execute a defined sequence of commands when you run the utility. To automate the ECSF Command Line Administration Utility in this way, you must configure the startup script to take inputs from a text file that you create.

The input file must contain one command per line. Any values that are not passed in with a command and for which you are typically prompted (for example, username and password) must occupy their own lines in the file. You must include a `connect` command since this is not passed in during the automated startup, and you must also include the `exit` command as the last command in the file to exit the ECSF Command Line Administration Utility. Example 28-1 illustrates a sample list of commands for an input file.

You must know all of the object IDs when you create the input file. Using the input file, you cannot create a new object and then manage it in one automation.

#### Example 28-1 Sample Input File

```plaintext
connect to database fusionhost123 1566 fh123
fusion
fusion
manage instance 1
set param "SES_ADMIN_SERVICE"="http://example.com:7777/search/api/admin/AdminService"
set param "SES_ADMIN_USERNAME" = "eqsys"
exit
```
28.2.9.1 How to Configure the Startup Script

To configure the startup script to automatically run the commands you defined in the input file, you must modify the startup script to include the AUTOMATE command in the STARTUP_PARAMS parameter.

1. Open the runCmdLineAdmin.bat (Windows) or runCmdLineAdmin.sh (Linux) script, located in ORACLE_HOME/jdeveloper/ecsf, in a text editor.

2. Set the STARTUP_PARAMS parameter to AUTOMATE and point to the location of input file. For example,

   \[\text{STARTUP_PARAMS} = \text{AUTOMATE /scratch/commands.txt}\]

   where commands.txt is the input filename.

3. Click Save.

   The output of the ECSF Command Line Administration Utility is displayed on the screen (or can be redirected), and errors are logged in the log file as usual. If the input file cannot be found, the ECSF Command Line Administration Utility runs in its usual mode and waits for the user to input a command through the prompt.

28.3 Set Up Oracle Enterprise Manager and Discovering ECSF

While you can use the ECSF Command Line Administration Utility to quickly test and manage the searchable objects, Oracle Enterprise Manager Fusion Applications Control is necessary to manage the life cycle of searchable objects in the production environment.

The ECSF runtime application needs to register its MBean to WebLogic's Domain Runtime MBean server, and the Oracle Enterprise Manager Fusion Applications Control needs to invoke all ECSF runtime operations through the MBean.

To access the Fusion Applications Control, you must install and configure Oracle Enterprise Manager (EM) to work with ECSF. You do not need to set up Oracle Enterprise Manager if you plan to use the ECSF Command Line Administration Utility to administer search.

To set up Oracle Enterprise Manager for ECSF, you must perform the following tasks:

1. Register the ECSF runtime application MBean.
2. Install Oracle Enterprise Manager.
3. Discover ECSF in Oracle Enterprise Manager.
4. Add users to the Administrators group.

Multiple developers can share one single Oracle Enterprise Manager application with the Fusion Applications Control.

28.3.1 How to Register the ECSF Runtime MBean to the Integrated WebLogic Server

The ECSF runtime application registers an MBean (oracle.ecsf.mbean.SearchRuntimeAdminMXBean) in WebLogic's Domain Runtime MBean server through a listener class (oracle.ecsf.mbean.RegisterMbeanContextListener). All ECSF runtime operations are invoked through the MBean.
To register the MBean, do the following:

1. Add the MBean listener to web.xml.
2. Create the application enterprise archive (EAR) file.
3. Configure the data sources.
4. Deploy the ECSF application.
5. Start the Oracle WebLogic Server instance.

Registering the ECSF runtime MBean to the Integrated WebLogic Server makes the MBean available to remote clients such as Fusion Applications Control in Oracle Enterprise Manager.

28.3.1.1 Add the MBean listener to web.xml

Add the MBean listener by modifying web.xml to include oracle.ecsf.mbean.RegisterMbeanContextListener.

To add the MBean listener do the following:

1. In the view-controller project in the Application Navigator, expand Web Content, then expand WEB-INF, and open web.xml.
2. Add the following <listener> element in web.xml:

   ```xml
   <listener>
     <listener-class>oracle.ecsf.mbean.RegisterMbeanContextListener</listener-class>
   </listener>
   ```
3. Save.

28.3.1.2 Creating the Application EAR File for Deployment

Create the application EAR file to be deployed. Right-click the application name and navigate to Deploy > ECSF application deployment profile > to EAR file. When the deployment is complete, you can find the generated EAR file in the JDeveloper log message window.

28.3.1.3 About Configuring Data Sources in Oracle WebLogic Server

You must configure data sources in Oracle WebLogic Server for MBean integration. Search for a data source with the JNDI name SearchDBDS. If any exist, make sure to look at the connection and validate that SearchDBDS is pointing to the correct database. If SearchDBDS is not listed, you must create a data source with jdbc/SearchDBDS as the JNDI name and with the connection information to the database against which the Fusion web application is running.

28.3.1.4 About Deploying the ECSF Application Using the EAR File

Make the MBean available by deploying the EAR file you created to Integrated WebLogic Server. Make sure that the EAR file is deployed and the application status is active in the final step.
28.3.1.5 About Starting the Oracle WebLogic Server Instance

When the MBean is available after you deploy the enterprise archive (EAR) file, you can start the Oracle WebLogic Server instance by selecting **Start Server Instance** from the **Run** menu.

28.3.2 About Installing Oracle Enterprise Manager

You must install Oracle Enterprise Manager to access the Fusion Applications Control. Installing Enterprise Manager allows you to then enable it to discover the ECSF custom application target in Oracle WebLogic Server.

28.3.3 How to Discover ECSF in Oracle Enterprise Manager

To use Fusion Applications Control in Oracle Enterprise Manager, you must first enable Oracle Enterprise Manager to discover the ECSF custom application target in Oracle WebLogic Server. When you discover ECSF in Oracle Enterprise Manager, you enable Oracle Enterprise Manager to detect and display the Fusion Applications Control. You only need to discover the ECSF custom application target in Oracle WebLogic Server once. When it is discovered, you can directly launch EM with the following URL:

http://your machine name:port/em

To discover ECSF in Oracle Enterprise Manager, do the following:

1. Invoke the target discovery page in Oracle WebLogic Server with the following URL:

   http://your machine name:port/em/faces/as/discovery/addWeblogic

2. Complete the following fields and click **Submit**:
   - **Host** (the name of the machine that hosts Oracle WebLogic Server)
   - **Port** (the Oracle WebLogic Server runtime port number, for example, 7101)
   - **Username** (for example, weblogic)
   - **Password** (for example, weblogic)

   **Note:**
   You can discover multiple Oracle WebLogic Servers by specifying a value for **Farm Name Prefix** (for example, Farm).

3. Click **Farm_DefaultDomain**.

4. At the Oracle Enterprise Manager login page (http://your machine name:port/em), log in with the following credentials:
   - Username: weblogic
   - Password: weblogic
5. To start Fusion Applications Control, navigate to Farm_DefaultDomain > Fusion Middleware > Enterprise Crawl and Search Framework and click EcsfRuntimeApp.

**Note:**
You only need to discover the ECSF custom application target in Oracle WebLogic Server once. When it is discovered, you can directly launch EM with the following URL:

```
http://your machine name:port/em
```

### 28.3.4 About Adding Users to the Administrators Group

To access the ECSF pages in Fusion Applications Control, users must be created and added to the Operator group and above on Oracle WebLogic Server.
This chapter describes how to create searchable objects in Oracle Fusion Applications. This chapter includes the following sections:

- Introduction to Creating Searchable Objects
- About Defining Searchable Objects
- About Securing Searchable Objects
- About Configuring Search Features
- About Configuring Custom Properties for Searchable Objects

29.1 Introduction to Creating Searchable Objects

Searchable objects are sets of data that make view objects available for full text search. They are used in an abstract way for exposing business data to search engines. For example, a purchase order as a searchable object would be defined as a set of searchable properties and its relationship to other searchable objects. Business data can be both structured and unstructured, such as data residing in a database, file attachments (including images), and documents.

The abstraction allows searchable objects to be bound to different contexts at runtime and to be described and used within that context. Because the binding information describes how a searchable object behaves in a given context, it is sometimes called search metadata.

To create searchable objects, you must perform the following tasks:

1. Define the searchable objects and searchable attributes.
2. (Optional) Enable the capability to crawl searchable objects with file attachments. For more information, see About Enabling Search on Fusion File Attachments.
3. (Optional) Enable the capability to crawl searchable objects with Oracle WebCenter tags. For more information, see About Enabling Search on WebCenter Tags.
4. Secure the searchable objects.
5. Configure the search features.
6. Configure the custom properties for the searchable objects.
About Defining Searchable Objects

29.2 About Defining Searchable Objects

Oracle Enterprise Crawl and Search Framework (ECSF) is used to integrate search functionality in Oracle Fusion Applications by defining searchable objects and its attributes. Defining the searchable objects enables the corresponding view objects and their attributes to search, and creates the necessary metadata for ECSF. The ECSF metadata can be packaged into an application archive and subsequently used by the ECSF runtime environment to deploy data sources into Oracle Secure Enterprise Search (Oracle SES) to perform crawl and index operations. All artifacts (for example, Java archive files, Oracle Application Development Framework objects, and so on), on which the view objects depend, must be packaged in the enterprise archive (EAR) file to make the searchable objects usable during runtime for both crawl and query operations.

Note:

You can also package ECSF metadata into metadata archive (MAR) files.

Consider the following when you define searchable objects and searchable attributes:

- Rather than reuse or functionally overload view objects designed for other purposes, construct view objects (parent and children) for search purposes (that is, include only attributes you intend to search).

- Exclude large objects (LOBs) from searchable objects unless you intend to search LOB contents.

- Restrict the use of multilevel view objects to five levels to avoid causing severe feed performance degradation.

- Collocate all servers (for example, ECSF, Oracle SES, database, and so on) and follow standard performance, scalability, and reliability network performance guidelines to minimize network latency between servers.

- Remember that feed throughput is directly proportional to the amount of data that is pulled from the database to be indexed.

- Be selective with the searchable objects to ensure that only a limited portion of data from the primary table is crawled to maximize crawling and indexing performance.

- Remember that attachments (Oracle Content Management SDK or LOBs) require record by record processing, and every record requires one additional network round-trip.

- Store attributes only when necessary. Use attributes stored in Oracle SES only for the following purposes:
  - Faceted navigation
  - Advanced search
  - Search result actions
  - Primary keys
If the value of an attribute must be searchable, then place it in the body of the searchable object.

- Use common attributes across Oracle Fusion Applications. Stored attributes common to all the searchable objects must share the same name and data type. For information about preventing conflicts, see What You May Need to Know About Preventing Search Attribute Naming Conflicts.

- Use the default security plug-in (oracle.ecsf.impl.DefaultSearchPlugin) as much as possible. Use of the default security plug-in ensures that the searchable objects are secured properly.

- Set the data type for stored attributes correctly and avoid conflicting attribute names.

- Identify the appropriate incremental crawl approach for your searchable object. Initial crawl refers to the first time data is crawled and indexed into a search engine. During initial crawl, all data intended to be crawled is sent to the search engine. All subsequent crawls are incremental crawls. During incremental crawls, only data that has been added or modified since the last crawl is sent to the search engine for indexing.

ECSF supports two mechanisms to identify newly added or modified data. You must choose and implement at least one of the following mechanisms:

- Configuring the Crawl Date Column searchable attribute property. For information, see How to Make View Object Attributes Searchable.

- Raising change events. For information, see About Raising Change Events Synchronously.

If neither mechanism is implemented, then the data that has been added or modified since the initial crawl will not be indexed, and therefore will not be searchable.

29.2.1 Understanding How to Define Searchable Objects

Before you can define searchable objects, you must be familiar with the following:

- Oracle JDeveloper and Oracle Application Development Framework technology. You must also understand how to build applications using entity objects, view objects, and view links.

- How view objects are used in ECSF. For more information, see the "Managing Search with Oracle Enterprise Crawl and Search Framework" chapter in the Administrator’s Guide.

- How to use Groovy expressions when defining search metadata. For information, see About Using Groovy Expressions in ECSF.

29.2.1.1 How to Define Searchable Objects and Attributes

To define searchable objects and searchable attributes, use the Search navigation tab of the overview editor in Oracle JDeveloper to complete the following tasks:

1. Make view objects searchable.

2. Make view object attributes searchable.
29.2.2 About Using Groovy Expressions in ECSF

ECSF indexes data based on a view object, which represents the top-level view object to crawl. Many business objects are hierarchical, and ECSF optimizes the Oracle Application Development Framework (Oracle ADF) methods of describing such hierarchies by using view links. ECSF supports multiple levels of a view hierarchy. By defining additional view objects and linking the top-level view object to the additional view objects through a view link, parent, child, grandchild, and so on relationships are formed. After a view link is set, you can reference data in those successive objects for indexing by using Groovy, a Java-like scripting language that is dynamically compiled and evaluated at runtime.

When you design your view object to be searchable, you must configure view links to generate only the Destination Accessor and not the Source Accessor.

When querying for root records during crawl time, ECSF also traverses the view link accessors to query child view objects. ECSF follows the parent view link accessors until it reaches the limit for the number of view accessor levels to be crawled. The default limit is set to 5 levels. If a parent view object has a view link accessor to its child view object, and the child view object also has a view link accessor pointing back to the parent view object, then the code cycles. It is not necessary to index the data in both parent and child view objects. When creating a view link object in Oracle JDeveloper, generate the view link accessor only in the parent view object and not in the child view object.

When defining search metadata, Groovy expressions are used to reference the attributes of view objects and the attributes of their child, grandchild, and so on view objects so that the attribute (string) values can be added into the ECSF index. You are required to enter Groovy expressions for fields such as Title, Body, and Keyword.

You can also use Groovy expressions to localize ECSF. For information, see About Localizing ECSF Artifacts.

29.2.2.1 About Referencing View Object Attributes as Variables

Groovy expressions can reference view object attributes (voAttrName) as variables. For example, you can enter the following Groovy expression in the Title field:

"Purchase Order " + RowId + " created for " + Customer + " on " + CreatedDate
Note:
View object attribute names are case-sensitive.

If the value for RowId is 1234, the value for Customer is ABC Inc, and the value for CreatedDate is 1/1/2007, then a search would return the following title value:

Purchase Order 1234 created for ABC Inc on 1/1/2007

The ECSF runtime code parses, then evaluates the expressions in the context of a view object, and returns the title with values for the variables.

Note:
To reference a stored view object attribute whose alias corresponds with an Oracle SES default search attribute (for example, Language), you must change the alias of the view object attribute to something other than any name of the Oracle SES default search attributes (for example, use alias Lang, instead of Language). Changing the alias prevents conflicts between Oracle SES and ECSF stored view object attributes. Alternatively, you can create view object transient attributes and reference them in the expressions. For more information, see What You May Need to Know about Preventing Conflicts with Default Search Attributes.

29.2.2.2 About Referencing Child View Object Attributes

Groovy expressions can also reference child view object attributes (viewLinkAccessorName.voAttrName) as variables by using the view link accessor names to access child attributes.

For example, you can enter the following Groovy expression in the Title field:

"Product Codes: " + ProductsView.ProdCode

Note:
View object attribute names are case-sensitive.

If the view object represented by the ProductsView view link accessor contains one record, and the value of its ProdCode attribute is 1XYZ, then a search would return the following title value:

Product Codes: 1XYZ

If a child view object referenced by an expression contains more than one record, then the values of all the child records are concatenated. For example, if the ProductsView view link accessor contains three records, and the values of its ProdCode attribute are 1XYZ, 2ABC, and 3STU, then a search would return the title value:

Product Codes: 1XYZ 2ABC 3STU.

The view link accessor name for a child view object is available on the view link that points to the child view object. The name of the view link accessor is the Destination value. In Figure 29-1, which shows the accessor information about the Relationship navigation tab of the view link editor, the name of the view link accessor is ZipLov.
You can also find the name of view link accessor in the `<ViewLinkAccessor>` tag of the XML source code of the view object.

Alternatively, you can use the `curdoc` keyword to access the current document, as shown in Example 29-1.

The `curdoc` keyword is used to access child documents and their attributes.

**Example 29-1 Using curdoc to Access Child Documents and Their Attributes**

```groovy
s = "";
if (curdoc.getChildDocs("ProductsView") != null) {
    for (d in curdoc.getChildDocs("ProductsView")) {
        s = s + d.attributes.ProdCode + " ";
    }
}
"Product Codes: " + s;
```

**29.2.2.3 About Referencing View Object Attributes in Multilevel Searchable Objects**

Groovy expressions can also reference view object attributes (`viewLinkAccessorName.voAttrName`, `viewLinkAccessorName.viewLinkAccessorName.voAttrName`, and so on) in searchable objects with a multilevel structure.

By default, the view link depth is set to 5. To index fewer or more than five levels of data in the view hierarchy, you must set the `oracle.ecsf.max.links.depth` property in system properties to the desired value. Figure 29-2 illustrates the Purchase Order searchable object with a five-level structure.
Figure 29-2 Searchable Object with Five Levels

Using Groovy expressions in the search metadata, you can access attributes in the levels PO Line Detail, Line Item Product, and Product Doc. For example, you can enter the following Groovy expression in the Title field:


If a grandchild view object referenced by an expression contains more than one record, then the values of all the grandchild records are concatenated. For example, if you reference view object attributes with viewLinkAccessorName.viewLinkAccessorName.voAttrName, then the result of the expression POLine.LineDetail.AttributeX is the concatenation of all AttributeX values for the LineDetail attribute of all POLine levels of the current PurchaseOrder searchable object.

29.2.2.4 About Formatting View Object Attribute Values

When writing Groovy expressions, it can be useful to generate strings that contain formatted versions of view object attribute values. For example, you may want to generate formatted strings such as "01/30/07" or "Jan 30, 2007" for an attribute value of type java.sql.Date.

To format attribute values, you must first determine the Java data types. The data types of view object attributes can vary. For example, some attributes may return simple strings, while others may return java.sql.Date, java.sql.Timestamp, or numbers like java.lang.Long.

One way to determine the types is to write a test Groovy expression that displays the type names. For example, if the view object has an attribute called Hiredate, then you can use the Groovy expression, as shown in Example 29-2, in the Body field.

The class name of the Hiredate attribute's value is displayed and can be viewed in the Data Feed:

Hiredate type: java.sql.Date.

After you determine the Java data type of a view object attribute, you can apply standard Java formatting techniques to format its value. Example 29-3 shows a sample Groovy expression for formatting a date.

The Groovy expression evaluates to:

Hire date: 01/30/2007

To format a number attribute named Qty of data type java.lang.Long using comma separators, write a Groovy expression, such as the one shown in Example 29-4.
The Groovy expression evaluates to:

*Quantity: 2,450*

**Example 29-2  Sample Groovy Expression to Determine Attribute’s Java Data Type**

'Hiredate type: ' + (Hiredate != null ? Hiredate.getClass().getName() : 'null');

**Example 29-3  Sample Groovy Expression for Formatting Dates**

fmt = new java.text.SimpleDateFormat('MM/dd/yyyy');  // create a date formatter (standard java class)
'Hire date: ' + fmt.format(Hiredate);

**Example 29-4  Sample Groovy Expression for Formatting Numbers**

fmt = new java.text.DecimalFormat('#,###,###');
'Quantity: ' + fmt.format(Qty);

### 29.2.3 What Happens When You Use Groovy Expressions in ECSF

The Groovy expressions are compiled and evaluated at runtime to display the desired string value in the crawl data feeds.

### 29.2.4 About Making View Objects Searchable

Use the Search page of the overview editor in JDeveloper, as shown in Figure 29-3, to set search property values for view objects.

**Figure 29-3  Search Page for Configuring Search Properties of View Objects**

- **Primary Table**: 
  - **Title**: 
  - **Body**: 
  - **Language Attribute**: 
  - **Search Plugin**: 
    - **Enable Real-Time Query Authorization**: 
    - **Secure View-Object Name**: 

- **Searchable Attributes**: 
  - **Search Result Actions**: 
  - **Facets**: 

**Note:**

The Search page is not editable if the view object or existing searchable object is read-only.
29.2.4.1 Set Search Property Values for View Objects

Make view objects searchable by setting search property values for them.

To set search property values for view objects, do the following:

---

**Caution:**

Do not modify the Oracle Fusion Applications Help searchable object named TopicSearchPVO.

---

1. In the overview editor, select the **Search** navigation tab.

2. Complete the **Primary Table** field by doing one of the following:
   - If the view object is based on an entity object, then select a value from the **Primary Table** dropdown menu.
   - If the view object is not based on an entity object, then enter the schema, table name, and table alias for the primary table. Use the format `DATABASE_SCHEMA.TABLENAME TABLE_ALIAS_NAME` (for example, `fusion.Emp Employee`).

You can either enter text directly into the text box or use the Select Primary Table dialog. For more information, see Using the Select Primary Table Dialog.

---

**Note:**

The table alias name you enter must match the table alias name used in the view object's SQL statement. You can view the SQL statement by selecting the view object's **Query** tab.

---

3. In the **Title** field, enter a Groovy expression to be evaluated to a string for the desired title of the search result. For more information, see About Using Groovy Expressions in ECSF.

The **Title** field allows for multiple lines of text.

You can also click the **Edit** icon to open the Edit Expression Editor, where you can enter Groovy expressions for the desired title of the search result. Click **OK** to save.

---

**Note:**

Do not include attributes of type character large object (CLOB) or binary large object (BLOB) in the Groovy expressions for the **Title**, **Body**, and **Keywords** fields, or you will receive an error. All columns with type CLOB or BLOB in a view object or its child view objects are processed as Oracle SES attachments.

---

4. In the **Body** field, enter Groovy expressions to provide additional information about the search results. This appears below the title. For more information, see About Using Groovy Expressions in ECSF.

You can also click the **Edit** icon to open the Edit Expression Editor, where you can enter Groovy expressions for the desired title of the search result. Click **OK** to save.
5. In the **Keywords** field, enter the keywords in the form of Groovy expressions. For more information, see About Using Groovy Expressions in ECSF.

You can also click the **Edit** icon to open the Edit Expression Editor, where you can enter Groovy expressions for the desired title of the search result. Click **OK** to save.

The values of the keywords are evaluated at crawl time using Groovy expressions, and sent to Oracle SES as part of the document for indexing. After they are indexed, the values of the keywords are searchable for the document with which they are associated.

6. Select the desired view object attribute from the **Language Attribute** dropdown menu to specify the language of this view object record. If the object has no language attribute, then leave it blank.

**Note:**

Currently, ECSF provides a way for you to specify if an attribute is a language field. ECSF will assume that the value of this field for each instance is the language for this instance. ECSF initially supports the Oracle Fusion Applications language code. If no language field is specified for a given searchable object, then ECSF uses the language preference of the crawler user.

7. Complete the **Search PlugIn** field by using the Search PlugIn dialog. For more information, see About Using the Search PlugIn Dialog.

8. Check the **Enable Real-Time query authorization** checkbox if you want the view object to use real-time authorization. The view object requires the proper Oracle Fusion Middleware Extensions for Applications (Applications Core) data security configuration, and ECSF will use the view-object query with security predicates from Applications Core data security to enforce query-time authorization at runtime.

9. (Optional) In the **Secure View Object Name** field, specify an alternative view object to use for authorization. ECSF will use this view object to get an authorization query with Applications Core security predicates.

10. Save the view object.

**Note:**

To save the changes made to the **Primary Table**, **Title**, **Body**, and **Keywords** fields, you must move the focus from those attributes (that is, click outside each field as you complete them).

**Note:**

If the view object is open in JDeveloper when you update the corresponding searchable object file (*view_object_name_ECSF.xml*), then you must close and reopen the view object to view the updates.
29.2.4.2 Using the Select Primary Table Dialog

Use the Select Primary Table dialog to specify the primary table for the searchable object if the view object is not based on an entity object.

To use the Select Primary Table dialog, do the following:

1. From the Search navigation tab, click the Edit icon next to the Primary Table field.

   The Select Primary Table dialog appears, as shown in Figure 29-4.

   **Figure 29-4  Select Primary Table Dialog**

2. Select the desired schema from the Database Schema dropdown menu.

3. With the Tables checkbox selected in the Object Type field, click Query to list the available tables.

4. Select the desired table name from the list under Available Objects.

5. Enter an alias for the primary table in the Table Alias field.

   **Note:**

   The table alias entered must match the table alias used in the view object’s SQL statement. You can view the SQL statement by selecting the view object’s Query tab.

6. Click OK.

   **Note:**

   You must select a primary table from the Available Objects list to save the changes.
29.2.4.3 About Using the Search Plugin Dialog

ECSF provides an extension model to allow you to extend ECSF functionality. Implement a search extension to implement any number of the following Java interfaces to extend or customize ECSF functionality:

- `oracle.ecsf.Securable`, used to implement a security extension
- `oracle.ecsf.PreIndexProcessor`, used to customize or manipulate data before it is sent to the search engine for indexing, such as for enabling advanced search on child objects (that is, attribute filtering)
- `oracle.ecsf.PostQueryProcessor`, used to customize results before search results are returned

Use the Search PlugIn dialog to specify the extension you want to use for the searchable object.

**Note:**

If you do not specify a search extension for your searchable object, then the default security extension is used. The default security extension requires you to identify a secure attribute. For information, see How to Make View Object Attributes Searchable.

**Note:**

If the search extension is in the Oracle WebLogic Server shared library, then the ECSF library (`ecsf.jar`) must be present in the shared library for ECSF to load the interface.

29.2.4.3.1 How to Use the Search Plugin Dialog

To use the dialog, do the following:

1. From the Search navigation tab, click the **Edit** icon next to the **Search PlugIn** field.

   The Search PlugIn dialog appears, as shown in Figure 29-5.
2. In the **PlugIn Class Name** field, enter the name of the custom Java class that implements the methods to determine security and resolve the URL.

**Note:**

If there is no plug-in defined in the searchable object, and you are not using a custom security extension, then this value is null and by default is set to `oracle.ecsf.impl.DefaultSearchPlugin`, which is used at runtime.

3. Optional: Enter a parameter name in the **Parameter Name** field and its corresponding value in the **Value** field, and click **Add**.

4. To update an existing parameter, select the desired parameter in the table, change its value, and then click **Update**.

5. To delete an existing parameter, select the desired parameter in the table and click **Delete**.

6. Click **OK** to save the changes.

### 29.2.5 What Happens When You Make View Objects Searchable

JDeveloper captures the search metadata for each view object and writes it to an external XML file (searchable object) for consumption by the runtime component. Each searchable object corresponds to a view object. The file naming convention is `view_object_name_ECSF.xml`, and the file is created in the same location as its corresponding view object.

**Note:**

Manually deleting an ECSF file removes the search functionality from the corresponding view object.
Caution:

Do not manually modify the contents of the `view_object_name_ECSF.xml` file. If you manually modify the search metadata in the XML file, then the changes appear in the editor window when it is closed and reopened in JDeveloper, but metadata changes are not validated. Instead, use the Search navigation tab of the overview editor in JDeveloper to modify the search metadata.

---

ECSF metadata can be packaged into an EAR file or a MAR file for consumption during crawl time and query time.

### 29.2.6 What You May Need to Know About Making View Objects Searchable

The searchable object reflects any change in the search metadata only after you save the view object. Until then, the changes are in memory. When you save the view object, the search metadata is saved to the searchable object. If there is no existing searchable object corresponding to the view object, then a new searchable object is created and stored in the same location as the view object.

In addition, if you rename or delete a view object with a corresponding searchable object, then the searchable object is likewise renamed or removed from the project.

---

### 29.2.7 About Making View Object Attributes Searchable

Making view object attributes searchable creates the necessary metadata for:

- Advanced search
- Faceted navigation
- Security
- Search result actions

**Note:**

The more attributes you make searchable, the larger the index becomes, which slows the performance of the queries.

---

Use the Search navigation tab of the overview editor in JDeveloper, shown in Figure 29-6, to set property values for view object attributes.
Using the Search navigation tab, you can perform the following tasks on view object attributes:

- Make view object attributes searchable.
- Modify searchable attributes.
- Delete searchable attributes.

### 29.2.7.1 How to Make View Object Attributes Searchable

Make view object attributes searchable by setting search property values for them.

1. In the overview editor, select the **Search** navigation tab.

2. Click the **Add** icon in the **Searchable Attributes** table header to open the Searchable Attribute Properties dialog, shown in Figure 29-7.

### Figure 29-7  Searchable Attribute Properties Dialog

3. Select the desired view object attribute from the **Attribute Name** dropdown menu.

4. Complete the remaining fields to define the property set for the searchable attribute. For information about the searchable attribute properties, see Table 29-1.
**Note:**

Before creating new stored attributes, check the list of Oracle SES attribute names and types to avoid conflicts. See *Oracle Fusion Applications Reference for Oracle SES Attributes*.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stored</td>
<td>Select this checkbox to store the view object attribute in Oracle SES as a separate custom search attribute. The view object attribute’s Alias property will be used as the name of the custom search attribute. The Alias property can be updated and renamed to avoid name and type conflicts. See <em>What You May Need to Know About Preventing Search Attribute Naming Conflicts</em>. The attributes stored in Oracle SES are used for Advanced Search, Actionable Results, and Faceted Navigation. For more information, see <em>About Defining Search Result Actions</em> and <em>How to Implement Faceted Navigation</em>. Selecting this checkbox enables the Weight field.</td>
</tr>
<tr>
<td>Secure Attribute</td>
<td>Select this checkbox to use the attribute’s value to secure the document at crawl time and to determine which users can access the indexed object at query time.</td>
</tr>
</tbody>
</table>
Table 29-1  (Cont.) Searchable Attribute Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crawl Date Column</td>
<td>Select this date column during incremental crawls to see if that record should be recrawled. Select this checkbox to enable the crawler to detect changes to the searchable object based on the date column. You must be able to reference the Crawl Date Column checkbox in a SQL predicate for ECSF to detect added or changed data. During incremental crawls, scheduled using either the Oracle Fusion Applications Control for ECSF or ECSF Command Line Administration Utility, only objects that have been modified since the last crawl are sent to Oracle SES. If the Crawl Date Column checkbox is specified for multiple searchable attributes, then all the date columns are used in the SQL query to retrieve data that has been modified since the last crawl. The SQL query uses the OR condition to get the incremental set of data. For example, if the Crawl Date Column checkbox is selected for CREATED_DATE and LAST_UPDATE_DATE in the searchable object, and the SQL query uses select * from EMP, then the following incremental crawl SQL query is generated: Select * from EMP where (CREATED_DATE between LAST_CRAWL_TIME and CURRENT_TIME) or (LAST_UPDATE_DATE between LAST_CRAWL_TIME and CURRENT_TIME) ECSF will check the values of all attributes in the Searchable View Object (SVO) that are marked as Crawl Date columns, find the most recent one, and use the value of that column as the LastModifiedDate stored attribute's value. In other words, ECSF will pick the most recent Crawl Date column value to use as the LastModifiedDate value. This allows users to perform queries using Advanced Search for documents that were modified within a certain date range.</td>
</tr>
<tr>
<td>Weight</td>
<td>Select the Stored checkbox to enable this field. Enter a value 1 to 10 (low to high), or select from the dropdown menu, to attach a weight to the stored attribute. Weights affect the ranking of the search results. If the weight is set to 1, then the stored attribute gets no boost. If the weight is set to 2 or 3, then the stored attribute gets very low boost (added to custom attribute Headline2). If the weight is set to 4 or 5, then the stored attribute gets low boost (added to custom attribute Headline1). If the weight is set to 6 or higher, then the stored attribute gets high boost (added to custom attribute Reference Text).</td>
</tr>
<tr>
<td>Override Source</td>
<td>Select this property to allow facets to be based on child view object attributes. This feature lets a stored attribute to be defined against a child view object attribute and a facet to be defined against this stored attribute. This allows users to filter search results so that only results whose child view object attribute matches a certain value will be returned. The Override Source property will support values of the format ViewLinkAccessorName.AttributeName.</td>
</tr>
</tbody>
</table>
Note:
Only stored and secured view object attributes are available for advanced search (that is, the Stored and Secure Attribute checkboxes are selected).

5. Click OK.
6. Save the view object.

Note:
If the view object is open in JDeveloper when you update the corresponding searchable object file (view_object_name_ECSF.xml), then you must close and reopen the view object to view the updates.

29.2.7.2 How to Modify Searchable Attributes
Modify a searchable attribute by editing the search property values for the view object attribute.

1. In the overview editor, select the Search navigation tab.
2. Expand the Searchable Attributes table header to expose the table of searchable attributes.
3. Select the searchable attribute you want to modify. This highlights the entire row.
4. Click the Edit icon in the Searchable Attributes table header to open the Searchable Attribute Properties dialog, shown in Figure 29-7.
   The attribute you selected on the Search navigation tab is displayed in the Attribute Name field.
5. Select or deselect the checkboxes to modify the property set for the searchable attribute. For information about the searchable attribute properties, see Table 29-1.
6. Click OK.
7. Save the view object.

29.2.7.3 How to Delete Searchable Attributes
Deleting the searchable attributes removes the search metadata for the view object attribute.

1. In the overview editor, select the Search navigation tab.
2. Expand the Searchable Attributes table header to expose the table of searchable attributes.
3. Select the searchable attribute you want to remove. This highlights the entire row.
4. Click the Delete (red X) icon in the Searchable Attributes table header.
   The attribute you selected on the Search navigation tab is removed from the table of searchable attributes.
5. Save the view object.

29.2.8 What Happens When You Define Searchable Attributes

JDeveloper captures the search metadata for each view object, including its attributes, and writes it to an external XML file (searchable object) for consumption by the runtime component. Each searchable object corresponds to a view object and includes the view object attributes. The file naming convention is `view_object_name_ECSF.xml`, and the file is created in the same location as its corresponding view object.

During crawl time, the ECSF runtime server uses the view attributes that are annotated for searching to construct documents for indexing.

**Caution:**

Do not manually modify the contents of the `view_object_name_ECSF.xml` file. If you manually modify the search metadata in the XML file, then the changes appear in the editor window when it is closed and reopened in JDeveloper, but metadata changes are not validated. Instead, use the Search navigation tab of the overview editor in JDeveloper to modify the search metadata.

ECSF implicitly adds the following attributes to Oracle SES indexes:

- **ECSF_SO_NAME.** This attribute stores the fully qualified searchable object name that corresponds to the searchable object on which the Oracle SES data source is based.

- **ECSF_TAGS.** This attribute is created if Oracle WebCenter Portal tags are associated with the searchable object. For information, see About Enabling Search on WebCenter Tags.

- **DEFAULT_ACL_KEY.** ECSF uses this attribute to store access control list (ACL) keys for the document.

29.2.9 What You May Need to Know About Defining Searchable Attributes

The searchable object reflects any change in the search metadata only after you save the view object. Until then, the changes are in memory. When you save the view object, the search metadata is saved to the searchable object. If there is no existing searchable object corresponding to the view object, then a new searchable object is created and stored in the same location as the view object.

In addition, if you rename or delete a view object with a corresponding searchable object, then the searchable object is likewise renamed or removed from the project.

**Note:**

Do not manually delete an attribute that has a corresponding search attribute from a view object, because it causes unexpected search results.
29.2.10 What You May Need to Know about Preventing Conflicts with Oracle SES
Default Search Attributes

Oracle SES supports system-defined default search attributes that may conflict with
ECSF stored view object attributes. For example, for Purchase Order 123 you define a
stored view object attribute with the alias Language and value US. However, Oracle
SES contains a default search attribute also named Language, but it has en as its
value. When you reference the view object attribute in a Groovy expression, such as
when you define a search result action of URL type where target="http://
example.com/q=dj&lang=\"+Language, you expect the search result action to
display as http://example.com/q=dj&lang=US.

However, the search result action displays as http://example.com/
q=dj&lang=en because the Oracle SES default search attribute value overrides the
value of the ECSF stored view object attribute of the same name.

Note:
The attribute conflict does not consider case. For example, a conflict still
occurs if the stored view object attribute’s alias is LANGUAGE (all caps) and the
Oracle SES default search attribute name is Language.

The following are the Oracle SES default search attributes:

- Author
- Description
- Headline1
- Headline2
- Headline3
- Host
- Infosource
- Infosource Path
- Keywords
- Language
- LastModifiedDate
- Mimetype
- Reference Text
- Subject
- Title
- Url
- Urldepth
Author, LastModifiedDate, and Subject are the exceptions, and can be used to enhance usability of the Oracle SES UI and to decrease the number of custom attributes in Oracle SES.

To prevent a conflict between Oracle SES default search attributes and ECSF stored view object attributes, you can either change the alias of the stored view object attribute to something other than any name of the Oracle SES default search attributes, or you can create a view object transient attribute and set it as a stored attribute, and then reference the transient attribute in the expressions.

To resolve the conflict in the given example, you can change the alias value of the Language view object attribute from Language (default) to Lang. The view object attribute alias is used to retrieve the value of the view object attribute in expressions.

Alternatively, you can resolve the conflict by creating a view object transient attribute, such as one named Lang, and use it in the expressions (for example, target="http://example.com/q=dj&lang="+Lang). By default, when the values of transient attributes are sent to Oracle SES, ECSF assigns the transient attribute a unique alias. When their values return as part of query results, they will not conflict with any default search attributes in Oracle SES. When expressions containing transient attributes are evaluated, ECSF converts the transient attribute names to the aliases and retrieves the data correctly.

29.2.11 What You May Need to Know About Preventing Search Attribute Naming Conflicts

Because Oracle SES supports facets only on attributes of type STRING, if a facet is defined on a nonstring attribute, then ECSF automatically converts the stored attribute type to a string before sending the attribute to Oracle SES.

However, a conflict may occur when a stored attribute of the same name is generated for a view object with no facets. Searchable attributes in Oracle SES are unique across the entire instance, so if multiple searchable objects contain the same attribute name of different types, then only the attribute (regardless of type) of the first searchable object crawled is used by Oracle SES. ECSF does check for stored attribute conflicts. For more information, see About Checking for Stored Attribute Conflicts.

Consider the following example:

You create a view object oracle.apps.crm.ustomer360.CustomerFVO with a set of attributes and types that are based on the underlying tables that use the Oracle ADF standard UI. The view object attribute contains the following information stored as Oracle ADF metadata:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Attribute Alias</th>
<th>Attribute Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>NAME</td>
<td>VARCHAR2</td>
</tr>
<tr>
<td>OrganizationId</td>
<td>ORGANIZATION_ID</td>
<td>NUMBER</td>
</tr>
<tr>
<td>Description</td>
<td>DESCRIPTION</td>
<td>VARCHAR2</td>
</tr>
</tbody>
</table>

You use the Search Designer to annotate a subset of these attributes (Name and OrganizationId) to store in Oracle SES for search purposes. For more information, see About Search Designer.
When Oracle SES crawls oracle.apps.crm.customer360.CustomerPVO, ECSF sends a document for each customer in the table. Each document contains the following attribute details:

<table>
<thead>
<tr>
<th>Attribute Alias</th>
<th>Attribute Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>VARCHAR2</td>
</tr>
<tr>
<td>ORGANIZATION_ID</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

In Oracle SES, NAME and ORGANIZATION_ID are created as customer attributes with the respective types. Due to the global nature of custom attributes in Oracle SES, if an attribute of the same name already exists, then no new attribute is created even if its type is different. Values for the attributes with conflicting name and type pairs are not stored in Oracle SES.

The issue surfaces when ORGANIZATION_ID is used as a facet (to enable users to narrow down the results per organization tree). ECSF implements a logic that detects whether an attribute is used for a facet. If it is used for a facet, then ECSF automatically changes the attribute type from NUMBER to VARCHAR2 because Oracle SES does not support facets on attributes with type NUMBER. This causes a conflict with the already existing ORGANIZATION_ID stored attribute of type NUMBER.

To prevent this conflict and allow Oracle SES to index both attributes, you must change the alias of the stored attribute that is used for facets. Go to the JDeveloper ADF view object attribute editor and update the Alias property value, as shown in Figure 29-8. For example, in the view object attribute editor, change the Alias value from PARTY_ID to PARTY_ID_FACET.
Before creating new stored attributes, check the list of Oracle SES attribute names and types to avoid conflicts. See *Oracle Fusion Applications Reference for Oracle SES Attributes*.

### 29.2.11.1 About Checking for Stored Attribute Conflicts

ECSF checks for stored attribute conflicts during Search Object deployment. If ECSF detects that the SO has attributes in it that will cause a conflict in SES, an error message will be shown that describes the error. Depending on the situation, the message will appear similar to one of these:

- The source attribute in runtime.EmpViewAdmin11gTest with name Host and type NUMBER conflicts with the attributes defined in these sources: Source: runtime.EmpView DataType: STRING Source: runtime.EmpView2 DataType: STRING

- The source attribute in runtime.EmpViewAdmin11gTest with name Mgr and type NUMBER conflicts with the attribute defined in SES with type STRING. This attribute is not being used by any sources.

You will have to change the SO so that the conflicting attribute either has a new name or has the same type as the attributes already defined in SES.

### 29.3 About Securing Searchable Objects

ECSF determines if a user has access to a search category depending on whether the user has permission to access the searchable objects in the category. *Search categories,*
also called search groups, are the logical collections of searchable objects that facilitate

group search on related items. Search categories are directly used for querying. If all of

the searchable objects in a search category are not accessible to the user, then that
category does not appear in the user’s category list. In this case, ECSF runtime does

not return that category when SearchCtrl.getSearchGroups() is called. However, if any one of the searchable objects in a search category is accessible to the user, then that category does appear in the user’s category list.

To secure searchable objects:

1. Set permissions for searchable objects
2. Create the security realm
3. Create the application policy store

### 29.3.1 How to Set Permissions for Searchable Objects

Set permissions for searchable objects by using the Search PlugIn dialog to enter the

permissions parameters. For information, see About Using the Search PlugIn Dialog.

To set permissions, do the following:

1. In the **PlugIn Class Name** field, enter the name of the custom Java class that
   implements the methods to determine security. If you are not using a custom
   security extension, enter oracle.ecsf.impl.DefaultSearchPlugin.

2. Add the following parameter names and values, and click Add for each name and
   value pair:
   - **Parameter Name:** FUNCTION_PERMISSION_NAME
     **Value:** PURCHASE_ORDER_VIEW_DETAILS
   - **Parameter Name:** FUNCTION_PERMISSION_ACTION
     **Value:** view
   - **Parameter Name:** FUNCTION_PERMISSION_CLASS
     **Value:** RegionPermission

   **Note:**

   The value of **FUNCTION_PERMISSION_NAME** and
   **FUNCTION_PERMISSION_ACTION** must be the same as the value of the
   permission name and action in the **jazn-data.xml** file.

3. Click OK.

The parameters are used to validate permissions in the ECSF security classes.
Searchable objects with no permissions set are accessible by all users.

### 29.3.2 How to Create the Security Realm

The user in the security realm is deployed to the Oracle WebLogic Server security

realm. Add a **jazn.com realm** to **jazn-data.xml**.

To add a security realm, do the following:
1. Expand Application Resources > Descriptors > META-INF, and open the jazn-data.xml file.


3. Navigate to jazn.com > Users, and click Add to add a user with the following information:
   
   Name: scott
   
   Credentials: weblogic

   The jazn-data.xml file is updated with the security realm, as shown in Example 29-5.

   The user Scott is associated to the jazn.com security realm.

4. Save.

   Example 29-5  jazn.com Security Realm

   <jazn-realm default="jazn.com">
   
   <realm>
   
   <name>jazn.com</name>
   
   <users>
   
   <user>
   
   <name>scott</name>
   
   <credentials>{903}O/XvB3XDx97MYp4sUSWwT3Q5KPLIEciA</credentials>
   
   </user>
   
   </users>
   
   </realm>
   
   </jazn-realm>

29.3.3 How to Create the Application Policy Store

The policy store is used to determined which users have access to which objects. Add an application policy store to jazn-data.xml.

To add the application policy store, do the following:

1. Expand Application Resources > Descriptors > META-INF, and open the jazn-data.xml file.

2. Select Application Policy Store, and click New to add a new policy store. Enter TestPermission for the display name.

3. Select Application Roles, and click Add to add a new role. Name it admin.

4. Select Application Roles, and shuttle the user scott from the Available section to the Selected section.

5. Select Application Policies, and click New to add a new policy. Enter View Orders for the display name.

6. Select the View Orders application policy, go to the Principals tab, and click Add to add a principal with the following information:

   Name: admin
   
   Class: oracle.security.jps.service.policystore.ApplicationRole
   
   Type: role
Leave the **Realm Name** field blank.

7. Go to the **Permissions** tab, and click **Add** to add a permission with the following information:

   Name: `PURCHASE_ORDER_VIEWDETAILS`
   Class: `oracle.adf.share.security.authorization.RegionPermission`
   Actions: `view`

   The `jazn-data.xml` file is updated with the application policy store, as shown in **Example 29-6**.

   The application policy store includes the values that you specified.

**Example 29-6    Application Policy Store**

```xml
<policy-store>
  <applications>
    <application>
      <name>TestPermission</name>
      <app-roles>
        <app-role>
          <name>admin</name>
          <class>oracle.security.jps.service.policystore.ApplicationRole</class>
          <members>
            <member>
              <name>scott</name>
              <class>oracle.security.jps.internal.core.principals.JpsXmlUserImpl</class>
            </member>
          </members>
        </app-role>
      </app-roles>
      <jazn-policy>
        <grant>
          <grantee>
            <display-name>
              View Orders
            </display-name>
            <principals>
              <principal>
                <type>
                  role
                </type>
                <class>oracle.security.jps.service.policystore.ApplicationRole</class>
                <name>admin</name>
              </principal>
            </principals>
            <permissions>
              <permission>
                <class>oracle.adf.share.security.authorization.RegionPermission</class>
                <name>PURCHASE_ORDER_VIEWDETAILS</name>
                <actions>view</actions>
              </permission>
            </permissions>
          </grant>
        </jazn-policy>
      </application>
    </applications>
  </policy-store>
```
29.4 About Configuring Search Features

In addition to basic and advanced search, ECSF allows you to further improve the search experience with the Actionable Results and Faceted Navigation search features, which you must configure.

---

**Note:**

No configuration is required for integrating Saved Search functionality.

---

To configure search features, you can complete the following tasks:

- Define search result actions.
- Implement faceted navigation.

29.4.1 About Defining Search Result Actions

Associating actions with the searchable objects and exposing the action links in the search results allows Oracle Fusion Applications users to run specific actions on a given search result. You can either define actions as URLs so the user can then either go to a specific web page related to the search result, or define actions as references to ADF task flow definitions so the user can then launch a specific task on the search result. For more information, see How to Add Dynamic Main Area and Regional Area Task Flows to a Page. Figure 29-9 illustrates an example of the results of a filtered search.
Clicking the title (default action URL) opens the Sonoma - Elites page. An additional action link allows the user to e-mail the owner.

You can perform the following tasks to define search result actions:

- Add search result actions
- Modify search result actions
- Delete search result actions

Use the Search navigation tab of the overview editor in JDeveloper, shown in Figure 29-3, to define actions for search results.

To define search result actions for Oracle Fusion Applications Search, you must complete additional configuration tasks. For information, see How to Use the Actionable Results API with Search.

29.4.1.1 About the Access URL

During runtime, the Access URL allows Oracle SES and the end user to access the applications from the Query page. The Access URL contains the view object name, the row’s primary keys, and the action name, the values of which are used to look up the action definition, query for the record, evaluate action parameters, and construct target URLs. The Access URL also points to the Redirect Service.
29.4.1.2 Using the Redirect Service

The Redirect Service is used for resolving URLs and redirecting users to the resolved URL. Invoking the Redirect Service improves performance when attributes referenced in the action definition not stored in the index.

The redirect logic includes the following steps following a request:

1. Use the view object name and action name to look up the action definition.
2. Use the primary key values to query for the row in the database.
   
   Attribute values are obtained from search attributes. If an action refers to unstored attributes, primary keys are used to obtain their values during redirect.
3. Construct an indexable document from the row.
4. Resolve the URL.
   
   For task action type:
   
   • Look up the task and construct a TaskFlowID.
   
   • Look up the task parameters, if there are any.
   
   • For each task parameter, evaluate the parameter value by passing the field and value map and the expression into an evaluator, add it to the resolved parameters list, then construct the URL using the same ControllerContext.getTaskFlowURL API.
   
   For URL action type:
   
   • Evaluate the parameterized URL by passing the field/value map and the expression into an evaluator.
5. Redirect the user to the resolved URL.

29.4.1.3 Add Search Result Actions

Add search result actions to associate actions to searchable objects. The action links you define display with the search results.

To add search result actions, do the following:

1. In the overview editor, select the Search navigation tab.
2. Click the Add icon in the Search Result Actions table header to open the Search Result Actions dialog, shown in Figure 29-10.
3. In the **Name** field, enter a name for the action.

**Note:**
No two actions can share the same name. This comparison is case insensitive.

4. From the **Action Type** dropdown menu, select **Task** (default) to specify a task to be performed on the search result or select **URL** to specify a web page.

**Note:**
Only bounded task flows can be launched through the URL mechanism.

5. Perform one of the following:

- If you chose **URL** for Action Type in Step 4, then in the **Action Target** field, define the action by entering a Groovy expression that generates the URL that is invoked when the user clicks the action.

  For URLs that point to an external site, you must configure the target expression to generate a fully qualified URL, which includes both the protocol (such as `http://`) and the host name (for example, `example.com`).

  For internal link URLs that point to pages on the same application from which the search is performed, configure the target expression to generate a relative URL (for example, `/EmployeeDetailPage?empId=\(+\)EmpId`). Relative URLs are invoked relative to the current host name and port number. This allows for the action to succeed on any application server on which the search is running.

  You can also click the **Edit** icon next to the **Action Target** field to use the Edit Expression Editor dialog for entering the Groovy expression. For more information, see About Using Groovy Expressions in ECSF.
The URL must be no longer than 32,000 in length. You can reference only stored attributes. For example, you can enter "http://www.example.com/search?hl=en&q=" + SRCompanyZip.

**Caution:**
Using unstored attributes results in an exception at query time when the action is resolved. Also, do not refer to child documents when defining the action definition.

- If you chose Task for Action Type in Step 4, then leave the **Action Target** field blank. However, you must define the **TaskName** and **TaskFile** properties in the Action definition. For information, see **Defining Properties for Bounded Task Flows**.

6. In the **Title** field, enter a Groovy expression to be evaluated to a string for the desired display title of the action as you would like it to appear on the Search Results page. For more information, see **About Using Groovy Expressions in ECSF**.

You can also click the **Edit** icon next to the **Title** field to use the Edit Expression Editor dialog for entering a Groovy expression.

7. Select the **Default Action** checkbox to make this action the default action to be performed on the search results.

**Note:**
You can set only one action as the default action.

8. For tasks, enter a parameter name in the **Parameter Name** field and its corresponding value in the **Value** field, then click **Add**.

Enter parameter values as Groovy expressions. You can use only stored attributes (for example, **SRNumber**) as parameters.

**Note:**
Using unstored attributes results in an exception at query time when the action is resolved. Also, do not refer to child documents when defining the action definition.

You can also click the **Edit** icon next to the **Value** field to use the Edit Expression Editor dialog for entering a Groovy expression.

Minimally, the **TaskName** and **TaskFile** parameters must be provided. The value of the **TaskName** parameter is a Groovy expression that returns the name of the task (that is, a name surrounded by double quotation marks). The value of the **TaskFile** parameter is a Groovy expression that returns the name of the task definition file. Values of other parameters are Groovy expressions that return the desired value. These parameters are passed into the task. For more information, see **About Using Groovy Expressions in ECSF**.

9. To update an existing parameter, select the desired parameter in the table, change its value, then click **Update**.
10. To delete an existing parameter, select the desired parameter in the table and click **Delete**.

11. Click **OK**.

12. To create additional search result actions, repeat Steps 2 to 11.

13. Save the view object.

### 29.4.1.4 Defining Properties for Bounded Task Flows

For bounded task flows, you must define the **TaskName** and **TaskFile** properties in the Action definition of the searchable object file (`view_object_name_ECSF.xml`). The task definition file, containing the value for **TaskName**, is usually located in the WEB-INF folder.

Edit the **TaskFile** parameter to point to the bounded task flow task definition XML file, located in the WEB-INF folder. For example, `\WEB-INF\filename`.

To define the **TaskName** and **TaskFile** properties, do the following:

1. Locate the task definition file and note its location and filename, for example, `\WEB-INF\task-flow-definition.xml`.

2. Open the task definition file, locate the `<task-flow-definition>` element, for example, `<task-flow-definition id='task-flow-definition'>` and note the value of the `id` attribute.

3. Open the searchable object file and locate the **TaskFile** parameter, then edit the value to reflect the location and filename of the task definition file, for example, `\WEB-INF\task-flow-definition.xml`.

4. Locate the **TaskName** parameter and edit the value to reflect the `id` attribute value of the `<task-flow-definition> element of the task definition file, for example, 'task-flow-definition'.

5. Save the searchable object file.

### 29.4.1.5 Modify Search Result Actions

You can modify search result actions as needed to change the action links that display with the search results.

To modify search result actions, do the following:

1. In the overview editor, select the **Search** navigation tab.

2. Expand the **Search Result Actions** table header to expose the table of actions.

3. Select the action you want to modify. This highlights the entire row.

4. Click the **Edit** icon in the **Search Result Actions** table header to open the Search Result Actions dialog, shown in Figure 29-10.

   The information of the action you selected is displayed.

5. Make the necessary changes to the desired fields. For more information about the fields, see **Adding Search Result Actions**.
6. Click OK.

7. Save the view object.

29.4.1.6 Delete Search Result Actions

You can delete search result actions to remove action links that display with the search results.

To delete search result actions, do the following:

1. In the overview editor, select the Search navigation tab.

2. Expand the Search Result Actions table header to expose the table of actions.

3. Select the action you want to delete. This highlights the entire row.

4. Click the Delete (red X) icon in the Search Result Actions table header.

   The action you selected on the Search navigation tab is removed from the table of actions.

5. Save the view object.

29.4.2 What Happens When You Define Search Result Actions

JDeveloper captures the search metadata for each view object, including the search result actions you define, and writes it to an external XML file (searchable object) for consumption by the runtime component. Each searchable object corresponds to a view object and includes the search result actions. The file naming convention is view_object_name_ECSF.xml, and the file is created in the same location as its corresponding view object.

Caution:

Do not manually modify the contents of the view_object_name_ECSF.xml file. If you manually modify the search metadata in the XML file, the changes appear in the editor window when it is closed and reopened in JDeveloper, but metadata changes are not validated. Instead, use the Search page of the overview editor in JDeveloper to modify the search metadata.

The search result actions you define during design time is parsed at runtime and appear in a table on the search results page.

29.4.3 What You May Need to Know About Defining Search Result Actions

The Name, Action Type, Action Target, and Title fields are required fields. You must input values for all three fields to save the action.
29.4.4 How to Implement Faceted Navigation

Facets are used to filter search results by attribute. A facet must point to an attribute that contains a list of values (LOV) definition. The LOV defines a way to get a list of values that make up the potential values for the attribute and can be used to filter results. To implement faceted navigation, you must perform the following tasks:

1. Create a searchable object and set it up for facets:
   a. Create a view object (base object) and identify the attributes you want to bind to facets.
   b. Create a view object (LOV object) for each of the attributes you identified in Step 1a to get a list of available values for the attributes.
   c. Create a view accessor on the base object for each LOV object to be used as a facet, and assign each LOV object to its corresponding attribute.
   d. Define the facet hierarchy. Since the faceted navigation path is hierarchical, you must form the structure so that the LOV objects form one facet definition. For each LOV object to be used as child, create a bind variable and view criteria.
      While defining the facet hierarchy, you can constrain the LOV object by the stored attribute.

2. Create the facets.

To illustrate the process of defining LOVs for facets, consider the following scenario: You want to create facet relationships for the EmpView base object. EmpView contains two stored attributes, StateID and CityID, on which you want to create facets. "City" is the child facet of "State." The values shown in the "City" facet are constrained by the value selected for "State." This scenario is used in the tasks listed in this section.

You can also configure stored transient attributes on the view object to define:

- Facets that support number or date ranges.
- Facets based on the values of multiple attributes.

Related Links

The following document provides more information about the topic discussed in this section:

- For more information, see the "Working with List of Values (LOV) in View Object Attributes" section in Developing Fusion Web Applications with Oracle Application Development Framework.

29.4.4.1 Defining Lists of Values

Defining LOVs for stored view object attributes creates facet relationships for the base object and its stored attributes. The following procedure uses the example of the EmpView base object and its two stored attributes, StateID and CityID.

To define list of values for stored view object attributes, do the following:

1. Define a view object (StatesView) to represent a list of values for a view attribute (StateID).
2. Define a view accessor for StatesView (StatesView1) on EmpView.

The view accessor lets you obtain the full list of possible values from the row set of the parent view object.

**Note:**
When you design your view object for search, make sure that you configure view links to generate only the Destination Accessor and not the Source Accessor.

3. In the Edit Attribute dialog for the stored attribute (StateID), select List of Values and select Enable List of Values.

4. Create a mapping from a child attribute to the stored attribute.

Values from the child attribute are used to filter the parent view object.

5. Specify which attribute on the view object contains the display value.
   - In the Attribute Editor, select List of Values and select Edit List UI Hints.
   - In the List UI Hints dialog, add a child attribute to display. The first attribute value is used as the display value.

6. Repeat Steps 1 to 5 to define a view object (CitiesView) to represent the stored attribute that is defined as the child facet.

**29.4.4.2 Constrain View Objects by Stored Attributes**

Constraining the CitiesView object by StateID makes sure that only the cities of the selected state are returned. Configure the CitiesView1 view accessor on EmpView so that the view criteria on CitiesView is applied when the user queries using CitiesView1. Based on the view criteria, only cities where its parent StateID is equal to the value of EmpView.StateId (that is, the currently selected state) is returned.

To constrain CitiesView by State, do the following:

1. Define a bind variable on CitiesView (StId).

2. Create a named view criteria (CitiesViewCriteria) on CitiesView.
   - On the View Criteria page of the View Criteria Editor dialog, click the Add Item button.
   - In the Criteria Items panel, define the criteria as follows:
     - Attribute: ParStateId
     - Operator: equal to
     - Operand: Bind Variable
     - Parameter: StId
     - Usage: Required
3. Click the Edit button to edit the view accessor (CitiesView1) for CitiesView on EmpView.

In the Edit View Accessor dialog, select the view criteria (CitiesViewCriteria) in the Available list and click the Move icon to add it to the Selected list, as shown in Figure 29-11.

**Figure 29-11   Edit View Accessor Dialog**

4. Map the StId bind variable on CitiesView to the StateId attribute of EmpView.

5. Ensure that the row-level bind values exist.

29.4.4.3 About Creating Search Facets

Faceted navigation allows Oracle Fusion Applications users to narrow their search results by setting filters, based on a set of predefined facets. For example, users can narrow their search results first by country, then by state, and then by city.

Search facets follow a tree structure. The root facet appears above the tree control, followed by child facets. All facets contain a name and an attribute, and can contain a child facet.

**Note:**

If you require both custom SQL query and facets for a view object, you must create the view object with the Updateable object through entity objects option and select Expert for Query Mode on the Query page.
You can define facets by creating search facets, modifying search facets, deleting root search facets, and deleting child search facets. Use the Search navigation tab of the overview editor in JDeveloper, shown in Figure 29-3, to define facets for faceted navigation.

You can create search facets to specify the root facet, facet name, an attribute, and child facets. Use the Facets dialog to add search facets.

Before you begin, define LOVs for stored view object attributes. For information, see Defining Lists of Values.

29.4.4.3.1 How to Add Search Facets

To add search facets, do the following:

1. In the overview editor, select the Search navigation tab.

2. Click the Add icon in the Facets table header to open the Facets dialog, shown in Figure 29-12.

   Figure 29-12 Facets Dialog

3. In the Facet field, enter a valid facet name for the root facet.

   Note:
   No two facets can share the same name at the searchable object level. This comparison is case insensitive.

4. Click the icon next to the Facet Display Name field to open and use the Select Text Resource dialog for selecting a text resource from an existing resource bundle, or
for creating and selecting a new text resource. For more information, see Using the Select Text Resource Dialog to Select a Matching Text Resource or Using the Select Text Resource Dialog to Create and Select a New Text Resource.

**Note:**
If no facet display name is specified, the label text of the view attribute corresponding to the facet is used as the display name. If there is no label text, then the view attribute name is used as the display name.

You can use resource bundles to localize ECSF. For information, see About Localizing ECSF Artifacts.

5. From the **Attribute Name** dropdown menu, select a searchable attribute for the root facet.

**Note:**
The Attribute Name dropdown menu lists only the searchable attributes whose `isStored` property is set to `true`. For information, see How to Make View Object Attributes Searchable.

No two facets can share the same attribute at the searchable object level.

6. To add a child facet, click **Add Facet**.

The new child facet is inserted below the selected facet in the tree structure. Any existing child facets are moved below it to the next level down in the structure.

7. Enter a name for the child facet in the **Facet** field.

**Note:**
No two facets can share the same name. This comparison is case insensitive.

8. Select an attribute for the child facet from the **Attribute Name** dropdown menu.

**Note:**
No two facets can share the same attribute.

9. Repeat Steps 5-7 to add more child facets.

10. Click **OK**.

11. Save the view object.

**29.4.4.4 About Defining a Facet to Use a Child View Object Attribute**

You can use this feature to define facets to filter result records against a child VO attribute. Without this feature, facets could only allow users to filter search results to those results where an attribute in the result record matches a certain value, such as `Color="Red"`, or `Status="Open"`. This feature allows a Stored Attribute to be defined
against a child VO attribute and for a facet to be defined against this stored attribute. Thus, users will be able to filter search results such that only results whose child VO attribute matches a certain value will be returned.

ECSF design time uses transient attributes to allow child VO attributes as searchable attributes by allowing them in the facet definitions.

The Override Source property value is expressed at the view attribute level and it is resolved at both the query time and crawl time. The Override Source drop-down list (see Figure 29-7) is populated with all the attribute names from the view links that are associated with the current VO.

The code for querying for facets will not change. At query time, a FacetPath will be converted to filters against the corresponding Stored Attributes in the facet path. With this feature, some stored attributes will contain multiple values, and the filter will select a search document if any of the values in its stored attribute matches the value in the filter.

Facet counts will work without any changes. When a stored attribute contains more than one value, each value will contribute to incrementing the count for that value.

29.4.4.4.1 Set Up a View Object with Support for Child Attributes in Facets

To set up a view object, do the following:

1. Identify the attributes you want to bind to facets in the current VO.

2. Identify the attributes you want to bind to facets from the child VO and create transient attributes on the current VO for each child attribute. (This assumes that you already established a relationship between parent and child VOs using View Link Accessors.)

3. Create a view object (LOV object) for each of the attributes you identified to create a list of available values for the attributes.

4. Create a view accessor on the base object for each LOV object to be used as a facet, and assign each LOV object to its corresponding attribute.

5. Create searchable attributes based on the attributes you created, including transient attributes. Select the Stored option for the attributes.

6. For the transient attributes, select the corresponding child view attribute name from the Override Source drop-down list in the Searchable Attributes popup. See Figure 29-7.

7. Define the facet hierarchy.

29.4.4.5 Using the Select Text Resource Dialog to Select a Matching Text Resource

Use the Select Text Resource dialog to select a text resource from an existing resource bundle.

To select a matching text resource, do the following:

1. From the Facets dialog, click the icon next to the Facet Display Name field to open the Select Text Resource dialog.

2. Select a resource bundle from the Resource Bundle dropdown menu.
Note:
If there is no resource bundle associated with the view object, then an external resource bundle is created in the application. Save the new resource bundle externally.

3. Enter values in the Display Value, Key, and Description fields to narrow down the list of matching text resources.
4. Select the desired text resource in the Matching Text Resources table.
5. Click Select. Click the Clear Selection button to clear your selection.

29.4.4.6 Using the Select Text Resource Dialog to Create and Select a New Text Resource

Use the Select Text Resource dialog to create and select a new text resource.
To create and select a new text resource, do the following:

1. From the Facets dialog, click the icon next to the Facet Display Name field to open the Select Text Resource dialog.
2. Select a resource bundle from the Resource Bundle dropdown menu.

Note:
If there is no resource bundle associated with the view object, then an external resource bundle is created in the application. Save the new resource bundle externally.

3. In the Display Value field, enter a string or any type of object to associate with the key in the page's resource bundle.
4. In the Key field, enter a string to uniquely identify a locale-specific object in the resource bundle.
5. In the Description field, enter a description of any length for the key and value pair.
6. Click Save and Select.

29.4.4.7 Modify Search Facets

You can modify existing search facets by changing the values of the facet attribute fields, adding child facets, or deleting child facets. Use the Facets dialog to modify search facets.
To modify search facets, do the following:

1. In the overview editor, select the Search navigation tab.
2. Expand the Facets table header to expose the table of root facets.
3. Select the root facet you want to modify. This highlights the entire row.
4. Click the Edit icon in the Facets table header to open the Facets dialog, shown in Figure 29-12.
5. In the tree structure, select the facet you want to modify.

6. Make your desired changes:
   
   • Change the values of the Facet Attributes fields. For more information about the fields, see About Creating Search Facets.
   
   • Add child facets. For more information, see About Creating Search Facets.
   
   • Delete child facets. For more information, see Deleting Child Search Facets.

7. Click OK.

8. Save the view object.

29.4.4.8 Delete Root Search Facets

You can delete root search facets to remove them from the table on the Search navigation tab.

To delete root search facets, do the following:

1. In the overview editor, select the Search navigation tab.

2. Expand the Facets table header to expose the table of root facets.

3. Select the root facet you want to remove. This highlights the entire row.

4. Click the Delete (red X) icon in the Facets table header.

   The root facet you selected on the Search navigation tab is removed from the table of root facets.

5. Save the view object.

29.4.4.9 Delete Child Search Facets

You can delete child search facets to remove them from the facet tree.

To delete child search facets, do the following:

1. In the overview editor, select the Search navigation tab.

2. Expand the Facets table header to expose the table of root facets.

3. Select the root facet you want to modify. This highlights the entire row.

4. Click the Edit icon in the Facets table header to open the Facets dialog, shown in Figure 29-12.

5. In the tree structure, select the facet you want to delete.

6. Click Delete.

7. Click Yes, when prompted with "Are you sure you want to delete the facets?"

   The facet you selected is removed from the tree.

8. Click OK.
9. Save the view object.

29.4.4.10 Define Facets That Support Ranges

You can define facets to support ranges for numbers and dates.

To define facets that support ranges, do the following:

1. Configure on the view object a transient attribute that summarizes the numeric or date attribute value into a range code. For example, create a transient attribute named `AmountRange` that contains the following Groovy expression for the number attribute:

   ```groovy
   if (Amount > 0 && Amount < 100) 'Range1'; else if (Amount >= 100 && Amount < 200) 'Range2'; else 'Range3';
   ```

   For a date attribute, you can create a transient attribute named `HireDate` that contains the following Groovy expression:

   ```groovy
   if (Hiredate.getYear() >= 80 && Hiredate.getYear() < 90) '80s'; else if (Hiredate.getYear() >= 91 && Hiredate.getYear() < 100) '90s'; else '2000s';
   ```

2. Define a static LOV with the range codes and display values.

3. Associate the LOV to the transient attribute.

4. Create a facet with the transient attribute in the facet definition.

29.4.4.11 Defining Derived Facets

You can define facets that are based on the values of multiple attributes. For example, `Project Status=Active if Status is not Closed AND Start Date is after Today`.

To define facets that support ranges, do the following:

1. Configure on the view object a transient attribute to classify a record into some code. For example, create a transient attribute named `Status` that contains the following Groovy expression for the number attribute:

   ```groovy
   if (Status == 'Active' && Closed != null) 'Active'; else 'Inactive';
   ```

2. Define a static LOV with the codes and display values.

3. Associate the LOV to the transient attribute.

4. Create a facet with the transient attribute in the facet definition.

29.4.5 What Happens When You Implement Faceted Navigation

JDeveloper captures the search metadata for each view object, including the search facets you define, and writes it to an external XML file (searchable object) for consumption by the runtime component. Each searchable object corresponds to a view object and includes the search facets. The file naming convention is `view_object_name_ECSF.xml`, and the file is created in the same location as its corresponding view object.

At runtime, the search interfaces provided by ECSF allow users to iterate through facets and further filter the query by selecting facet values.
Caution:
Do not manually modify the contents of the view_object_name_ECSF.xml file. If you manually modify the search metadata in the XML file, the changes appear in the editor window when it is closed and reopened in JDeveloper, but metadata changes are not validated. Instead, use the Search navigation tab of the overview editor in JDeveloper to modify the search metadata.

29.4.6 What You May Need to Know About Implementing Faceted Navigation
Facets can only be defined on attributes in the parent or top-level view object because only attributes on the parent or top-level view object can be created as index attributes in the underlying Oracle SES index. For example, if you want to create an “address” facet consisting of the following tree, Country > State > City > Zip Code, all four attributes (country, state, city, zip code) must be view object attributes in the parent or top-level view object. Attributes that are used for facets must exist on the parent view object, not on a child view object linked to the parent through a view link. If any of the information is in a child attribute, then the attribute must be joined into the parent view object.

Since facets can filter against only a single search category, faceted navigation is not supported with federated search.

29.5 About Configuring Custom Properties for Searchable Objects
You can configure custom properties for searchable objects to modify default runtime behavior or to make searchable object public.

29.5.1 How to Modify Default Runtime Behavior of Searchable Objects
The following custom properties can be set for searchable objects through the overview editor to modify default runtime behavior:

- oracle.ecsf.crawl.batch.size
- oracle.ecsf.crawl.datafeed.size
- oracle.ecsf.max.links.depth
- oracle.ecsf.split.mode
- oracle.ecsf.split.threshold

These properties, described in Table 29-1, can also be set as system parameters to apply the values to all searchable objects. For information, see How to Modify the Run Configuration of the View-Controller Project.

To configure customer properties for searchable objects, do the following:

1. In the Application Navigator, open the desired view object.
2. In the overview editor, select the General navigation tab.
3. Expand the Custom Properties section.
4. Add property name and value pairs. For more information about the properties, see Table 29-1.
29.5.2 About Making Searchable Objects Public

Making searchable objects public allows users to perform searches without needing to log in first. Public data sources do not require any user authentication and can support anonymous users.

To make a searchable object public, define the custom property `oracle.ecsf.searchableobject.public`, as shown in Example 29-7, in the `view_object_name.xml` file.

When this value is set to true, the Security attribute values for anonymous user property of the data source deployed to Oracle SES from this searchable object is set to the ACL value or values retrieved from the searchable object's plug-in class.

**Example 29-7  Custom Property for Making Searchable Objects Public**

```xml
<Properties>
  <CustomProperties>
    <Property
      Name="oracle.ecsf.searchableobject.public"
      Value="true"/>
  </CustomProperties>
</Properties>
```
This chapter describes how to configure security for ECSF.

This chapter includes the following sections:

- Introduction to Configuring ECSF Security
- Securing ECSF Credentials
- About Authorizing Users for Search Feeds
- About Securing the Searchable Application Data

30.1 Introduction to Configuring ECSF Security

ECSF secures credentials and searchable application data. The credentials are required for the ECSF engine to communicate with Oracle Secure Enterprise Search (Oracle SES) administration service, Oracle SES query service, and ECSF Security Service. ECSF also uses Secure Socket Layer (SSL) to secure the connections through which the credentials are transmitted. ECSF stores the credentials in the Credential Store Framework (CSF) of the Oracle WebLogic Server domain.

Configure the HTTP protocol to restrict the maximum post and message size to prevent denial-of-service (DoS) attacks, which makes the servlets unavailable.

30.2 Securing ECSF Credentials

Passwords are stored in the Credential Store Framework (CSF) of the Oracle WebLogic Server domain. These passwords are used to perform secure interaction between the ECSF engine and the Oracle SES server.

30.2.1 About Adding the Permission Policy

When the ECSF Runtime Server or the ECSF Client library is added to the projects in Oracle JDeveloper, the permission policy, shown in Example 30-1, is automatically added within the <jazn-policy> tag of the application's jazn-data.xml file located in src/META-INF.

Credentials are stored under mapName oracle.ecsf, oracle.apps.security, and oracle.wsm.security with a key in the format: username#engineInstanceId (for example, scott#1, where scott is the user on engine instance 1).

When the application is deployed, the policies in jazn-data.xml are merged into the system-jazn-data.xml file in WebLogic_domain/config/fmwconfig on Oracle WebLogic Server.
Note:

The following security deployment options for the application must be configured in JDeveloper for the policies to merge: policies, credentials, and users/groups.

Make sure that the policy migrates to the target Oracle WebLogic Server domain.

Related Links

The following document provides more information about the topic discussed in this section:

- For more information, see the *Oracle Fusion Middleware Applications Securing Applications with Oracle Platform Security Services* guide.

**Example 30-1 Permission Policy**

```xml
<grant>
  <grantee>
    <codesource>
      <url>file:${domain.home}/servers/${weblogic.Name}/tmp/_WL_user/oracle.ecsf/-</url>
    </codesource>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
      <name>context=SYSTEM,mapName=oracle.ecsf,keyName=*</name>
      <actions>*</actions>
    </permission>
    <permission>
      <class>oracle.security.jps.JpsPermission</class>
      <name>IdentityAssertion</name>
      <actions>execute</actions>
    </permission>
  </permissions>
</grant>
```

### 30.2.2 How to Configure Application Identities for Search

Oracle Fusion Applications include six search-related application identities that are seeded and are stored in the identity store:

- `FUSION_APPS_CRM_SES_CRAWL_APPID`
- `FUSION_APPS_FSCM_SES_CRAWL_APPID`
- `FUSION_APPS_HCM_SES_CRAWL_APPID`
- `FUSION_APPS_CRM_ECSF_SEARCH_APPID`
- `FUSION_APPS_FSCM_ECSF_SEARCH_APPID`
- `FUSION_APPS_HCM_ECSF_SEARCH_APPID`
- `FUSION_APPS_ECSF_SES_ADMIN_APPID`

Each pair of application identities, one pair for each product family, are used to integrate ECSF with Oracle Fusion Applications. The Credential Store Framework (CSF) stores the credentials to access the identities.
However, if you are developing applications on the Integrated WebLogi
instance, then you must manually configure the application identities to inte
ECSF for the crawl users: SES_ADMIN_USERNAME, SES_QUERY_PROXY_USERNAME,
and ECSF_SECURITY_USERNAME.

Note:
To prevent duplication of crawls, crawling and indexing of searchable object
data into Oracle SES must be performed by one crawler user. The single
crawler user, specified in the search engine instance parameter
ECSF_SECURITY_USERNAME, must have access to all searchable object data to
be indexed.
The required setup of a user depends on the application setup and is not
controlled by ECSF. For example, Oracle Fusion Applications includes three
application IDs that are created for crawling data:
FUSION_APPS_CRM_SES_CRAWL_APPID,
FUSION_APPS_FSCM_SES_CRAWL_APPID, and
FUSION_APPS_HCM_SES_CRAWL_APPID. You must make sure that the
proper roles, permissions, privileges, and so on are granted to the three
application IDs so they have access to the data to be crawled.

To configure the application identities, you must complete the following tasks:

1. Make sure the SearchContext is set to FusionSearchContextImpl.
2. Create the application identities.
3. Make sure the permission policies for the identity store and the JPS
   IdentityAssertion API are added to the jazn-data.xml file.

30.2.2.1 About Setting the SearchContext to FusionSearchContextImpl
In order for ECSF to handle the application identities of Oracle Fusion applications,
the SearchContext must be set to FusionSearchContextImpl. The SearchContext is
automatically set at runtime based on the runtime environment. If the SearchContext
is not set properly, then set the context using the oracle.ecsf.context system
property, for example:

-Doracle.ecsf.context='oracle.ecsf.fusion.FusionSearchContextImpl
'

For more information, see How to Modify the Run Configuration of the View-
Controller Project.

30.2.2.2 About Creating the Application Identities
Each of the crawl users (SES_ADMIN_USERNAME, SES_QUERY_PROXY_USERNAME,
and ECSF_SECURITY_USERNAME) must correspond to an application identity. Use
Oracle Enterprise Manager Fusion Applications Control for ECSF to set the crawl user
names and their corresponding passwords. For information, see the “Managing Search
with Oracle Enterprise Crawl and Search Framework” chapter in the Administrator’s
Guide.

For example, set the user names for Oracle Fusion Customer Relationship
Management to:
When the user names are set, you can update the corresponding password parameters for those users to the key names for the application identities. The format of the key name is `fullAPPID-KEY`.

This creates entries in the cwallet with the correct map/key pairs for the users.

### 30.2.2.3 About Adding the Permission Policy for the Application Identities

In order for ECSF to read and write to the application identity maps in the keystore and access the JPS Identity Assertion API, permissions must be granted to the three crawl users. The permission policies, shown in Example 30-2, are seeded in the `jazn-data.xml` file for Oracle Fusion applications and can be managed in Fusion Applications Control.

The permissions allow ECSF to read and write credential store entries that are not part of the `oracle.ecsf` map.

#### Example 30-2 Permission Policies for Application Identities

```xml
<grant>
  <grantee>
    <codesource>
      <url>file:${domain.home}/servers/${weblogic.Name}/tmp/_WL_user/oracle.ecsf/-</url>
    </codesource>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
      <name>context=SYSTEM,mapName=oracle.ecsf,keyName=*</name>
      <actions>*</actions>
    </permission>
    <permission>
      <class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
      <name>context=SYSTEM,mapName=oracle.wsm.security,keyName=FUSION_APPS_FSCM_ECSF_SEARCH_APPID-KEY</name>
      <actions>*</actions>
    </permission>
    <permission>
      <class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
      <name>context=SYSTEM,mapName=oracle.wsm.security,keyName=FUSION_APPS_HCM_ECSF_SEARCH_APPID-KEY</name>
      <actions>*</actions>
    </permission>
    <permission>
      <class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
      <name>context=SYSTEM,mapName=oracle.wsm.security,keyName=FUSION_APPS_CRM_ECSF_SEARCH_APPID-KEY</name>
      <actions>*</actions>
    </permission>
    <permission>
      <class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
      <name>context=SYSTEM,mapName=oracle.apps.security,keyName=FUSION_APPS_CRM_SES_CRAWL_APPID-KEY</name>
      <actions>*</actions>
    </permission>
  </permissions>
</grant>
```
30.3 About Authorizing Users for Search Feeds

New grants are needed to authorize users for the search feeds. You must manually update the application's `jazn-data.xml` file located in `src/META-INF` to enable authorization for users. Add the grant, shown in Example 30-3, inside the `<application>` section in the `<jazn-policy>` section.

The grantee should be the users or roles that you want to authorize to use the search feeds, as shown in Example 30-4.

The example shows how `jazn-data.xml` is modified to grant the permission to a role.

**Example 30-3  Grant for Search Feeds User Authorization**

```xml
<permission>
  <class>oracle.adf.share.security.authorization.MethodPermission</class>
  <name>ECSF_All_Services</name>
  <actions>execute</actions>
</permission>
```

**Example 30-4  Granting Permission to a Role**

```xml
<grant>
  <grantee>
    <principals>
      <principal>
        <class>oracle.security.jps.service.policystore.ApplicationRole</class>
        <name>AuthorizedUserRole</name>
      </principal>
    </principals>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.adf.share.security.authorization.MethodPermission</class>
      <name>ECSF_All_Services</name>
      <actions>execute</actions>
    </permission>
  </permissions>
</grant>
```
30.4 About Securing the Searchable Application Data

ECSF secures the searchable application data by authenticating and authorizing users who use ECSF to perform searches.

30.4.1 How to Secure the Searchable Application Data

Secure the searchable application data by enabling the use of the security model for authenticating and authorizing users.

To enable the use of the security model, do the following:

1. Create users in Oracle WebLogic Server. The user credentials are stored in Oracle WebLogic Server and can be used for authentication and authorization to query Oracle SES.

2. Create a separate user and add it to the Operators group to assign that user the Oracle WebLogic Server security role of Operator to obtain execute privileges on ECSF MBean operations.

3. Create an ECSF query proxy user. For more information, see the "Managing Search with Oracle Enterprise Crawl and Search Framework" chapter in the Administrator’s Guide.

4. Set the search engine instance parameters SES_QUERY_PROXY_USERNAME and SES_QUERY_PROXY_PASSWORD. For more information, see the "Managing Search with Oracle Enterprise Crawl and Search Framework" chapter in the Administrator’s Guide.

Note:

ECSF also supports allowing authenticated users to search business objects with no security policies attached to them.
This chapter describes how to validate and test search metadata. This chapter includes the following sections:

- Introduction to Validating and Testing Search Metadata
- About Validating the Search Metadata
- Testing Searchable Objects Through a Web Browser

### 31.1 Introduction to Validating and Testing Search Metadata

Search metadata is dependent on view object metadata, which is used in such areas as title, body expression, and searchable attributes. However, changes made to the view object metadata are not automatically reflected in the search metadata. To identify those changes, you must validate the search metadata by using the Validate button in the Search page of the Oracle JDeveloper overview editor for Oracle Enterprise Crawl and Search Framework (ECSF).

Testing the searchable objects ensures that they can be registered in the Oracle Fusion Applications database without issues.

### 31.2 About Validating the Search Metadata

To make sure that you are creating correct and valid search metadata on view object metadata, you must validate the metadata before applying changes. When you run validation on the search metadata, ECSF checks for the following:

- Table and alias of the primary table exists in the view object.
- Title, body, and keywords expressions are valid.
- Search extension is valid.
- Search attributes are valid, and all the search attributes exist in the view object.
- Search actions are valid:
  - Search action names are unique.
  - Default action exists. Warning is displayed when it does not exist.
  - Title expression is valid.
- Search facets are valid:
  - Facet name is unique.
31.2.1 How to Validate Search Metadata

Use the Search navigation tab of the overview editor in JDeveloper, shown in Figure 29-3, to validate search metadata. If you identify any errors (changes to the view object metadata that are not reflected in the search metadata), you must manually fix them.

To validate search metadata through the Validate button, do the following:

1. In the overview editor, select the Search navigation tab.

2. Click the Validate button.

If validation errors occur, the Search Validation Results dialog, shown in Figure 31-1, appears.

Figure 31-1 Search Validations Dialog

![Search Validation Results](image)

The dialog shows a list of validation errors.

3. Click OK when you finish viewing the errors.

31.3 Testing Searchable Objects Through a Web Browser

Before you register searchable objects in the Oracle Fusion Applications database, you should test the searchable objects by testing the Config Feed, Control Feed, and Data Feed.

ECSF prohibits multiple feeds per searchable object, so after achieving the desired results for the Config Feed, Control Feed, and Data Feed, you must reset the state of the feeds.

31.3.1 Perform Prerequisite Steps

Before you begin testing searchable objects, do the following:

1. Set the following Java option in the Run configuration dialog to test the searchable objects through a web browser: 

```java
-Doracle.ecsf.crawl.mode.debug=true
```

For more information, see How to Modify the Run Configuration of the View-Controller Project.
2. Set up ECSF security. For information, see Configuring ECSF Security.

3. Create searchable objects. For information, see Creating Searchable Objects.

4. Run the ECSF feed servlet. For information, see How to Run the ECSF Feed Servlet.

### 31.3.2 How to Run the ECSF Feed Servlet

Running the ECSF feed servlet in debug mode provides the servlet access to HTTP GET, the method that allows you to input URLs with arguments directly into the browser.

The web.xml file is automatically updated with the ECSF feed servlet, feed servlet mapping, and filter mappings as shown in Example 31-1 when the ECSF Runtime Server library is added to the project.

To ensure that the ECSF feed servlet runs properly, do the following:

1. Set the Application Navigator to display the Java archive (JAR) files and libraries by clicking the Navigator Display Options button in the Projects panel and selecting Show Libraries, as shown in Figure 31-2.

#### Figure 31-2 Navigator Display Options

2. In the Application Navigator, expand the ECSF Runtime Server library.

3. Expand the oracle.ecsf.feed package.

4. Right-click the SearchFeedServlet.class file and select Run to start the Integrated WebLogic Server instance. A browser opens to the following feed URL:

   http://localhost:7101/approot/searchfeedservlet/%2A
You can click the Terminate (red square) button to stop the ECSF feed servlet.

**Example 31-1  ECSF Feed Servlet, Feed Servlet Mapping, and Filter Mappings**

<servlet>
  <servlet-name>SearchFeedServlet</servlet-name>
  <servlet-class>oracle.ecsf.feed.SearchFeedServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>SearchFeedServlet</servlet-name>
  <url-pattern>/searchfeedservlet/*</url-pattern>
</servlet-mapping>

<filter-mapping>
  <filter-name>JpsFilter</filter-name>
  <servlet-name>SearchFeedServlet</servlet-name>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>REQUEST</dispatcher>
  <dispatcher>INCLUDE</dispatcher>
</filter-mapping>

<filter-mapping>
  <filter-name>ServletADFFilter</filter-name>
  <servlet-name>SearchFeedServlet</servlet-name>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>REQUEST</dispatcher>
</filter-mapping>

31.3.3 How to Test the Config Feed

To test the Config Feed, run the ECSF feed servlet with a modified URL.

1. Change the feed URL http://localhost:7101/approot/searchfeedservlet/%2A by:
   - Replacing localhost with the server name of the IP address.
   - Replacing %2A with the fully qualified name of the view object, including the package path.
   - Appending /ConfigFeed to the end of the URL.

   For example, if JDeveloper is running on the Linux server wllserver.com, and the fully qualified package path of EmpVO is oracle.ecsf.EmpVO (case-sensitive), the resulting Config Feed URL would be http://wllserver.com:8988/approot/searchfeedservlet/oracle.ecsf.EmpVO/ConfigFeed.

2. Refresh the web page.

   The resulting RSS feed, the Config Feed, should resemble the feed in Example 31-2.

   If the RSS feed does not appear, then either the runtime server is not set up properly or the path to the view object is incorrect. The URL is case-sensitive.
If no attribute exists for the `<securityAttribute>` tag, you must mark at least one searchable attribute as a Secure Attribute. For information, see About Making View Objects Searchable.

3. After adding the metadata, make sure to restart the Integrated WebLogic Server instance.

Example 31-2  Sample Results of the Config Feed

```xml
<rsscrawler xmlns="http://xmlns.example.com/search/rsscrawlerconfig">
  <feedLocation>http://localhost:8988/approot/searchfeedservlet/runtime.EmpView/ControlFeed</feedLocation>
  <feedType>controlFeed</feedType>
  <errorFeedLocation>/tmp</errorFeedLocation>
  <securityType>attributeBased</securityType>
  <securityAttribute name="DEPTNO" grant="true" />
</rsscrawler>
```

31.3.4 How to Test the Control Feed

To test the Control Feed, run the ECSF feed servlet with a modified URL.

1. Locate the URL provided in the `<feedLocation>` tag of the Config Feed. For information, see How to Test the Config Feed.

   In the Config Feed shown in Example 31-2, the URL is `http://localhost:8988/approot/searchfeedservlet/runtime.EmpView/ControlFeed`.

2. Change the feed URL by replacing localhost with the server name of the IP address.

   For example, if JDeveloper is running on the Linux server `example.com`, the resulting Control Feed URL would be `http://example.com:8988/approot/searchfeedservlet/runtime.EmpView/ControlFeed`.

3. Refresh the web page.

4. At the login screen, enter the username and password.

   The resulting RSS feed, the Control Feed, should resemble the feed in Example 31-3.

Example 31-3  Sample Results of the Control Feed

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<rss xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0">
  <channel>
    <title>Control feed for ECSF</title>
    <link>http://example.com:8988/approot/searchfeedservlet/runtime.EmpView/ControlFeed</link>
    <description>Control feed for Enterprise Crawl and Search Framework</description>
    <lastBuildDate>2008-04-06T19:20:08.159Z</lastBuildDate>
    <channelDesc xmlns="http://xmlns.example.com/orarss">
      <feedType>control</feedType>
    </channelDesc>
    <item>
      <link>http://example.com:8988/approot/searchfeedservlet/runtime.EmpView/DataFeed?Testing Searchable Objects Through a Web Browser
Validate and Test Search Metadata  31-5
```
31.3.5 How to Test the Data Feed

To test the Data Feed, run the ECSF feed servlet with a modified URL.

1. Locate the URL provided in the <link> tag under the <item> tag of the Control Feed. For information, see How to Test the Control Feed.

   In the Control Feed shown in Example 31-3, the URL is http://example.com:8988/approot/searchfeedservlet/runtime.EmpView/DataFeed?tableName=Emp&workUnitStart=AAHz%2BdADeAAA07UAAA&workUnitEnd=AAHz%2BdADeAAA07UAAV&type=ROWID.

2. Refresh the web page with the URL.

3. At the login screen, enter the username and password.

   The resulting RSS feed, the Data Feed, should resemble the feed in Example 31-4.

Example 31-4 Sample Results of the Data Feed

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<rss xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0">
  <channel>
    <title>RSS for Oracle Applications Search</title>
    <link>http://example.com:8988/approot/searchfeedservlet/runtime.EmpView/DataFeed</link>
    <description>RSS for Oracle Applications Search</description>
    <lastBuildDate>2008-04-06T19:36:08.950Z</lastBuildDate>
    <channelDesc xmlns="http://xmlns.example.com/orarss">
      <feedType>incremental</feedType>
    </channelDesc>
    <item>
      <link><![CDATA[http://ecsf.example.com/search/runtime.EmpView?EMPNO=7839]]></link>
      <title><![CDATA[KING: 5000]]></title>
      <itemDesc xmlns="http://xmlns.example.com/orarss">
        Employee department job salary data  KING
      </itemDesc>
    </item>
  </channel>
</rss>
```
### 31.3.6 How to Reset the State of the Feeds

Resetting the feed for the searchable object recrawls the object. Reset the state of the feeds by modifying the Config Feed URL.

1. Recall the Config Feed URL of the searchable object. For information, see [How to Test the Config Feed](#).
2. Append `?forceInitialCrawl=true` to the Config Feed URL, for example, `http://example.com:8988/approot/searchfeedservlet/runtim.eEmpView/ConfigFeed?forceInitialCrawl=true`
3. Paste the resulting URL in your browser and refresh the web page.
This chapter describes how to deploy searchable objects to the Oracle Enterprise Crawl and Search Framework (ECSF) application.

This chapter includes the following sections:

- Introduction to Deploying and Crawling Searchable Objects
- Deploying Searchable Objects and Dependencies
- About Crawling Searchable Objects

32.1 Introduction to Deploying and Crawling Searchable Objects

The ECSF application must include the searchable objects before you deploy it to the application server.

32.2 Deploy Searchable Objects and Dependencies

Searchable objects and their dependencies must be deployed as part of the ECSF application’s data model and user interface projects (Model and ViewController respectively) to make the searchable objects available for search. To deploy searchable objects, you must complete the following tasks:

1. Deploy the ECSF shared library to Oracle WebLogic Server.
2. Create an application.
3. If desired, change the application name and context root of the view-controller project.
4. Modify the run configuration of the view-controller project.
5. Add the ECSF Runtime Server library and the required Java archive (JAR) files to Model and ViewController.

32.2.1 About Deploying the ECSF Shared Library to Oracle WebLogic Server

The ECSF shared library eliminates the need for ECSF libraries to be packaged into each application. Instead, applications that depend on ECSF libraries can reference the ECSF shared library that is deployed to the Oracle WebLogic Server. The ECSF shared library contains the following Java archive (JAR) files:

- ecsf.jar
- search_admin_wsclient.jar
- search_client.jar
• ses_admin_ows_proxy.jar
• soap.jar

The ECSF extension in JDeveloper controls the reference to the ECSF shared library that is deployed to Oracle WebLogic Server. When you add the ECSF Runtime Server or ECSF Client library to a project, the reference to the ECSF shared library, shown in Example 32-1, is automatically added to the WebLogic deployment descriptor file (weblogic-application.xml).

The ECSF shared library is oracle.ecsf.

If the weblogic-application.xml file does not exist, one is created and updated with the reference to the ECSF shared library.

In addition, when you deploy an application to a Oracle WebLogic Server instance and the project contains ECSF libraries, the code checks the descriptor for the ECSF shared library reference. If no ECSF shared library reference is detected in the descriptor, one is added. The WebLogic deployment descriptor also contains a list of library dependencies for the application to be deployed to the Oracle WebLogic Server instance.

The ECSF shared library is automatically deployed to the Integrated WebLogic Server instance by the ECSF extension in JDeveloper through the JDeveloper Application Development Runtime Service (ADRS). However, you must manually deploy the ECSF shared library to the standalone WebLogic Server.

Example 32-1  Reference to the ECSF Shared Library

```xml
<library-ref>
  <library-name>oracle.ecsf</library-name>
</library-ref>
```

32.2.1.1 Update the SearchDB Data Source

The ECSF shared library is automatically deployed to the Integrated WebLogic Server instance by the ECSF extension in JDeveloper through the JDeveloper Application Development Runtime Service (ADRS). When the Integrated WebLogic Server instance is first started and the ECSF shared library is automatically deployed to it, the ECSF shared library creates a SearchDB data source in the Oracle WebLogic Server domain. The data source initially contains placeholder database connection information. You must manually update the data source after the Integrated WebLogic Server instance is started to include the correct connection information.

To update the SearchDB data source, do the following:

1. In the Domain Structure tree of the Oracle WebLogic Server Administration Console, navigate to Services, then JDBC, then Data Sources.
2. On the Summary of Data Sources page, click the data source name SearchDB.
3. Click the Connection Pool tab.
4. On the Connection Properties page, replace the default values for both the Connection URL and Properties boxes with valid connection values.
5. Click Save.
32.2.1.2 Deploy the ECSF Shared Library to the Standalone WebLogic Server Instance

You must manually deploy the ECSF shared library to the standalone WebLogic Server instance. The ECSF shared library creates a SearchDB data source in the Oracle WebLogic Server domain. During the process of deploying the ECSF shared library, you must provide the database connection information for the SearchDB data source, which is deployed with the shared library.

To deploy the ECSF shared library, do the following:

1. Extend the Oracle WebLogic Server domain by using the Oracle Fusion Middleware Configuration Wizard.
   For a Linux OS, execute:
   ```bash
   $WL_HOME/common/bin/config.sh
   ```
   For a Windows OS, execute:
   ```bat
   %WL_HOME%\common\bin\config.bat
   ```

2. Select the ECSF Shared Library Extension template (oracle.ecsf_11.1.1_template.jar), which is located in oracle/jdeveloper/common/templates/applications.

3. Configure the SearchDB data source by providing valid values for the following fields:
   - DBMS Host
   - DBMS Port
   - SID
   - Username
   - User Password

4. When a new version of the ECSF shared library is available, you must redeploy it.
   a. Remove the old oracle.ecsf library.
   b. Install the library enterprise archive (EAR) file (oracle/jdeveloper/ecsflibraries/oracle.ecsf_11.1.1/oracle.ecsf.ear) with the name set as oracle.ecsf.

Note:
You can also redeploy the ECSF shared library by using the Oracle Fusion Middleware Configuration Wizard.

32.2.2 How to Create an Application

Creating the application creates Model and ViewController, which must include the searchable objects and their dependencies.

To create a new application, do the following:
1. From the File menu, select New.
2. In the New Gallery dialog, select the General category and select Applications.
3. Select the Fusion Web Application (ADF) template and click OK.
4. In the Create Fusion Web Application (ADF) dialog, enter a name and location for the application in the Application Name and Directory fields.
5. Enter a value in the Application Package Prefix field.
6. Click Finish.

### 32.2.3 How to Change the Application Name and Context Root of the View-Controller Project

If desired, change the application name and context root of the view-controller project by modifying the Java EE application settings.

To change the Java EE application settings, do the following:

1. In the Application Navigator, right-click the view-controller project and select Project Properties.
2. In the Project Properties dialog, select Java EE Application in the left panel.
3. Change the value of the Java EE Web Application Name field to EcsfApp.
4. Change the value of the Java EE Web Context Root field to approot.
5. Click OK.

   The view-controller application name is set to EcsfApp, and the context root is set to approot.

### 32.2.4 How to Modify the Run Configuration of the View-Controller Project

Modify the run configuration of the view-controller project to run ECSF in debug mode.

To modify the run configuration, do the following:

1. In the Application Navigator, right-click the view-controller project and select Project Properties.
2. In the Project Properties dialog, select Run/Debug/Profile in the left panel.
3. Select Default in the Run Configurations list, then click the Edit button.
4. In the Edit Run Configuration dialog, select Launch Settings in the left panel.
5. If desired, enter additional parameter values in the Java Options field. Table 32-1 lists the ECSF system parameters. Separate each parameter with a space.
Note:
The parameters in Table 32-1 can be set in two ways:

- Add an entry to a file named `ecsf.properties`. If this file is found in the classpath by ECSF code, the property values in that file will be used. Entries in the file should be of the format:
  
  `property=value`

- Set the parameters using the Java System Properties. For example, add `-Dproperty=value` when starting the JVM.

If a property is found in both the `ecsf.properties` file and the Java System Properties, the value in the Java System Properties will be used. In other words, the Java System Properties have higher precedence.

These parameters values can also be specified at the searchable object level. For information, see About Configuring Custom Properties for Searchable Objects.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Java Command-Line Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oracle.ecsf.admin.dataobject.synchinterval</td>
<td><code>-Doracle.ecsf.admin.dataobject.synchinterval=n</code></td>
<td>Defines the interval (n), in milliseconds, for synchronizing the ECSF metadata in the cache with the ECSF metadata in the database. The default value is 30000 (30 seconds).</td>
</tr>
<tr>
<td>oracle.ecsf.connection.name</td>
<td><code>-Doracle.ecsf.connection.name=ConnectionName</code></td>
<td>Specifies the name of the database connection to be used. If not specified, SearchDB is used.</td>
</tr>
<tr>
<td>oracle.ecsf.context</td>
<td><code>-Doracle.ecsf.context=co</code></td>
<td>Defines the ECSF context to be used. The value can be set to <code>oracle.ecsf.fusion.FusionSearchContextImpl</code> for Oracle Fusion Applications; otherwise, the default ECSF context is used. The Oracle Fusion Applications search context handles Fusion specific details such as application identities.</td>
</tr>
<tr>
<td>oracle.ecsf.crawl.batch.size</td>
<td><code>-Doracle.ecsf.crawl.batch.size=n</code></td>
<td>Defines the data batch size within a data feed. The value determines the number of database rows (n) that are processed per batch within a data feed. The default size is 200.</td>
</tr>
<tr>
<td>oracle.ecsf.crawl.datafeed.size</td>
<td><code>-Doracle.ecsf.crawl.datafeed.size=n</code></td>
<td>Defines the size for the data feed. The value determines the number of documents (n) per data feed. The default size is 1000.</td>
</tr>
<tr>
<td>oracle.ecsf.datasource.name</td>
<td><code>-Doracle.ecsf.datasource.name=DataSourceName</code></td>
<td>Specifies the Java Database Connectivity (JDBC) data source name, such as <code>jdbc/SearchDBDS</code>.</td>
</tr>
<tr>
<td>Parameter Name</td>
<td>Java Command-Line Entry</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>oracle.ecsf.datesplitter.mode</td>
<td>-</td>
<td>Defines the algorithm used to split the records in the table since the last crawled time. The value can be set to DateRowId or DateOnly. DateRowId is the default value. In DateRowId mode, it takes the records since the last crawled time and splits into evenly sized workunits using rowid ranges. In DateOnly mode, it splits into workunits based on date ranges only. This may result in workunits with too many records if there are many updated records within a short amount of time. In that case, using the DateRowId mode is recommended.</td>
</tr>
<tr>
<td>oracle.ecsf.maxfacetsdefvalues</td>
<td>-</td>
<td>Specifies the maximum number of values (n) that a facet may contain. ECSF does not return the values if a facet (for example, Country) contains several values (for example, USA, Canada, and so on) that exceeds the maximum. The default value is 1000.</td>
</tr>
<tr>
<td>oracle.ecsf.maxlinks.depth</td>
<td>-</td>
<td>Specifies the maximum number of view object hierarchy levels to limit the depth of search.</td>
</tr>
<tr>
<td>oracle.ecsf.service.ws.timeout</td>
<td>-</td>
<td>Specifies the ECSF web service invocation timeout (n) in milliseconds. The default value is 90000 (90 seconds). Increasing the timeout value can resolve issues related to a slow system environment.</td>
</tr>
<tr>
<td>oracle.ecsf.split.mode</td>
<td>-</td>
<td>Defines batching strategy. When the model is simple, no batching is used. Use for simple testing. The key is defined in RelationalCrawlerImpl. Its value can be db, simple, or nothing. This value defines how the crawler splits the database table. If specified with value simple, then SimpleSplitter is used. If specified with value db (default) or nothing (value is blank, or the parameter is not included at all), then RowIdSplitter is used.</td>
</tr>
<tr>
<td>oracle.ecsf.split.threshold</td>
<td>-</td>
<td>Sets the splitting algorithm threshold to the percentage you specify (n). If the percentage of records returned by the searchable object SQL query versus the total number of records in the searchable object's primary table is less than the threshold percentage, then the view object RowIdSplitter algorithm is used. Otherwise, the default RowIdSplitter algorithm is used.</td>
</tr>
<tr>
<td>oracle.ecsf.cache.expireseconds</td>
<td>-</td>
<td>Defines the time (n), in seconds, for items to expire from the cache. The default value is 1800 (30 minutes).</td>
</tr>
</tbody>
</table>
Table 32-1  (Cont.) ECSF System Parameters

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Java Command-Line Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oracle.ecsf.cache.expireseconds.ObjectCache</td>
<td>- Doracle.ecsf.cache.expireseconds.ObjectCache=n</td>
<td>Defines the time (n), in seconds, for items to expire from the Searchable Objects cache. The default value is 24 hours.</td>
</tr>
<tr>
<td>oracle.ecsf.cache.expireseconds.FacetDefsCache</td>
<td>- Doracle.ecsf.cache.expireseconds.FacetDefsCache=n</td>
<td>Defines the time (n), in seconds, for items to expire from the Facet Definitions cache. The default value is 24 hours.</td>
</tr>
<tr>
<td>oracle.ecsf.cache.expireseconds.GroupCache</td>
<td>- Doracle.ecsf.cache.expireseconds.GroupCache=n</td>
<td>Defines the time (n), in seconds, for items to expire from the Searchable Group cache. This cache caches a Searchable Group using Engine Instance ID and Searchable Group name. The default is the value set by oracle.ecsf.cache.expireseconds.</td>
</tr>
<tr>
<td>oracle.ecsf.cache.expireseconds.GroupsCache</td>
<td>- Doracle.ecsf.cache.expireseconds.GroupsCache=n</td>
<td>Defines the time (n), in seconds, for items to expire from the Searchable Groups cache. This cache caches Searchable Groups using Engine Instance ID. The default is the value set in oracle.ecsf.cache.expireseconds.</td>
</tr>
<tr>
<td>oracle.ecsf.cache.expireseconds.EngineInstancesCache</td>
<td>- Doracle.ecsf.cache.expireseconds.EngineInstancesCache=n</td>
<td>Defines the time (n), in seconds, for items to expire from the Search Engine Instances cache. The default is the value set in oracle.ecsf.cache.expireseconds.</td>
</tr>
</tbody>
</table>

6. Click OK.

The run configuration is set to debug mode, and other ECSF system parameters are set.

32.2.5 About Adding the ECSF Runtime Server Library and Required Java Archive Files to the Model and View-Controller Projects

You must add the ECSF Runtime Server library and one of the following sets of required Java archive (JAR) files to both the Model and view-controller projects:

- For using ECSF for crawling and querying
  - oracle/wlserver_10.3/server/lib/wls-api.jar

- For using ECSF for querying only
  - oracle/jdeveloper/webservices/lib/soap.jar
Caution:

Make sure that `cwallet.sso` and `jazn-data.xml` are part of your application before adding the Java archive (JAR) files. You can do so through the Application Navigator by navigating to Application Resources, then Descriptors, then META-INF. The `cwallet.sso` file is created when you create a database connection. To create `jazn-data.xml`, right-click the META-INF folder select New Oracle Deployment Descriptor, select `jazn-data.xml`, and click Finish.

You do not need to add the Java archive (JAR) files that are included in a library that you have already added.

### 32.2.6 About Deploying the ECSF Application

After you update the Model and view-controller projects to include the searchable objects and dependencies, you must deploy the ECSF application. For information, see Setting Up Your JDeveloper Application Workspace and Projects.

### 32.3 About Crawling Searchable Objects

Make sure that the Oracle Secure Enterprise Search (Oracle SES) engine successfully crawls the searchable objects in the Oracle Fusion applications and indexes them as documents.

#### 32.3.1 How to Verify the Crawl

Use the Oracle Enterprise Manager Fusion Applications Control and Oracle SES administration user interface to verify the crawl.

**Note:**

The feed servlet must be running for Oracle SES to successfully crawl the data.

To verify the crawl, do the following:

1. Deploy the index schedule.
   
   For more information, see "Task 4: Deploy the Index Schedules" in Administrator’s Guide.

2. Start the index schedule.
   
   For more information, see "Task 5: Start the Index Schedules" in Administrator’s Guide.

3. Use the Oracle SES administration user interface to inspect whether the crawls were successful and verify how much data is crawled.
This chapter provides information on advanced topics for Oracle Enterprise Crawl and Search Framework (ECSF), including enabling and managing search, and troubleshooting ECSF.

This chapter includes the following sections:

- Introduction to Advanced Topics for ECSF
- About Enabling Search on Fusion File Attachments
- About Enabling Search on WebCenter Tags
- About Enabling Search on Tree-Structure-based Source Systems
- About Managing Recent Searches
- Setting Up Federated Search
- About Federating Instances
- About Raising Change Events Synchronously
- About Using the External ECSF Web Service for Integration
- About Localizing ECSF Artifacts
- About Using ECSF Diagnostics
- About Troubleshooting ECSF

### 33.1 Introduction to Advanced Topics for ECSF

ECSF offers additional functionality to enhance the search experience. In addition to search on business objects, ECSF supports search on Fusion file attachments, WebCenter tags, and tree structure-based source systems. ECSF also allows you to set up federated search so that users can search across Oracle Fusion Applications product families or across multiple Oracle Secure Enterprise Search (Oracle SES) instances.

Advanced topics also include using the external ECSF web service to integrate ECSF with Oracle Fusion Applications, implementing Coherence cache for search, localizing ECSF artifacts for international users, and information for troubleshooting ECSF.

### 33.2 About Enabling Search on Fusion File Attachments

ECSF supports the capability to crawl Oracle Fusion Applications file attachments that are associated with ECSF searchable objects and stored in the Oracle WebCenter Content repository.
33.2.1 How to Make File Attachments Crawlable

References to files in the content repository are stored in a special database table and are retrieved by using a view object named `oracle.apps.fnd.appcore.attachments.uiModel.view.AttachmentsVO`. Using a view link, you can make the `AttachmentsVO` a child of another view object. If a searchable object has a child `AttachmentsVO`, then ECSF automatically makes sure that the attachments are crawled when the searchable object is crawled.

At the time the searchable object is crawled, ECSF includes a content link in the document that is sent to Oracle SES for each attachment. When Oracle SES receives the data feed and finds the content link, it calls back to ECSF to retrieve the content of the attachment. ECSF then invokes an application programming interface (API) method that retrieves the attachment content from the content server and returns it to Oracle SES. Oracle SES indexes the searchable object and attachment content as one item.

The API handles authentication and authorization.

To make file attachments crawlable, do the following:

1. Add the Applications Core (Attachments) library to your project.
   
   This adds the `Attachment-Model.jar` and `Common-Model.jar` files, located in `JDEV_HOME\jdeveloper\jdev\oaext\adflib`, to your projects.

2. Create a view link between the searchable object and the `AttachmentsVO` to make `AttachmentsVO` a child of the searchable object.

   **Note:**
   
   When you design your view object for search, make sure that you configure view links to generate only the Destination Accessor and not the Source Accessor.

3. On the searchable object, define a view link accessor that points to the view link. For information, see How to Create Attachment View Links.

33.3 About Enabling Search on WebCenter Tags

ECSF supports the capability to crawl searchable objects with Oracle WebCenter Portal tags so that tags can be used as keywords or filters for search in Oracle Fusion Applications. A tag is a meaningful term attached to an object. Tags can be used for various purposes such as categorization, to-dos, and priorities.

**Note:**

ECSF does not support search using private tags.

Tags are single words stored as space-separated strings in the application database and are retrieved by using a view object called `TagSVO` (service view object). You must create a view link to make `TagSVO` a child object. At crawl time, the view link is used to locate `TagSVO`. Figure 33-1 illustrates crawl time with tags.
Purchase Order 123 has two tags, *Computer* and *Dell*. ECSF adds the tags for each record of the searchable object to the indexable document before the document is sent to Oracle SES for indexing. ECSF also creates a reserved attribute (of type string) called *ECSF_TAGS* to store tags in Oracle SES.

At query time, users can specify tag values as keywords or as filters. When tag values are input as keywords, the tag value is treated as a query string and returns results that include the objects with the specified tag. When tag values are used as filters, the tags are added to *QueryMetaData* and the query is run with filters on *ECSF_TAGS*. Only the objects with the specified tags are returned. *Figure 33-2* and *Figure 33-3* illustrates the difference between query time without a tag and query time with a tag.

**Note:**

Tags cannot be searched as individual entities.
In Figure 33-2, the indexed document for Purchase Order 123 contains two tags, Computer and Dell. The query on John returns all the documents that contain John. The results display the title, body, and all tags for the documents.
In Figure 33-3, query on the Purchase Order John and tag Dell returns only the documents that contain John AND the Dell tag. The results display the title, body, and all tags for the documents. If both Dell and Computer were specified as tags, then the query would return only the documents that contain both tags (that is, Dell AND Computer). You cannot specify tags using the OR condition (that is, Dell OR Computer), so the query cannot return documents that contain either the Dell tag or the Computer tag.

Enabling search on WebCenter Portal tags allows tags to be added to indexable documents and stored in the reserved attribute called ECSF_TAGS in Oracle SES. Tags can then be used as keywords or filters for search.

33.3.1 How to Enable Search on WebCenter Portal Tags

Perform the following tasks to enable search on WebCenter Portal tags:

1. Create a view link between the searchable object and the TagSVO. For information, see How to Implement Tags in Search.

   **Note:**

   When you design your view object for search, make sure that you configure view links to generate only the Destination Accessor and not the Source Accessor.

   The view link must use the accessor name TagSVO so that the search extension can navigate to the child object when it crawls.
2. Add tags to the indexable document.
3. Add tags to the query.

### 33.3.2 How to Customize Search on WebCenter Portal Tags

You can also perform the following tasks to customize search on WebCenter Portal tags:

1. Modify the tags in the indexable document.
2. Register change listeners.

### 33.3.3 How to Add Tags to Indexable Documents

To crawl tags, you must use Tag APIs to add tags to indexable documents. You can use Tag APIs in the search extension code.

To add tags, do the following:

1. Create a search extension that extends `DefaultSearchPlugin`, implements `PreIndexProcessor`, and includes a method called `preIndexProcess` that adds tags to some objects.

   **Note:**
   If you place the search extension in the Oracle WebLogic Server shared library, then the ECSF library (`ecsf.jar`) must be present in the shared library in order for ECSF to load the `PreIndexProcessor` interface.

2. Redeploy and crawl the objects.

#### Example 33-1  Sample Code for Adding Tags to Indexable Documents

```java
public class runtime.TestPlugin extends DefaultSearchPlugin implements PreIndexProcessor {
    public void preIndexProcess(SearchContext ctx, List<IndexableDocument> documents) {
        for (IndexableDocument doc : documents) {
            Object ename = doc.getFieldValue("ENAME");
            if (ename != null && "Zebra".equalsIgnoreCase(ename.toString())) {
                doc.addTags(new String[] { "Black", "White", "Stripes" });
                //doc.getTags and doc.clearTags can also be used here
                for (String tag : doc.getTags()) {
                    system.err.println(tag); // print out tag to stand err
                }
            }
        }
    }
}
```

You can implement code, such as the sample code shown in `Example 33-1`, in the search extension to extend `DefaultSearchPlugin`.

This extension adds three tags (`Black`, `White`, and `Stripes`) to a user named `Zebra`. 
33.3.4 About Adding Tags for Querying

After tags are crawled into Oracle SES, you can perform keyword searches on tags or filter on tags. QueryMetaData accepts tags for querying. When one or more tags are added to QueryMetaData, the query runs with filters on ECSF_TAGS.

Example 33-2 illustrates how tags are used for querying.

Adding tags for querying forces the query to find results where indexed documents contain the added tags. When more than one tag is added, the resulting documents must contain both tags. Documents containing only one of the tags are not returned.

Query SES retrieves tags all the time. The searcher adds tags to IndexedDocument by using setTags. In case of null in the attribute, no tags are added. IndexedDocument.getTags returns all the tags in the document.

Example 33-2 Sample API for Querying With Tags

```java
public void tagTest()
{
    SearchCtrl searchCtrl = new SearchCtrl();
    SearchHits searchHits = null;
    SearchContext searchContext = null;
    QueryMetaDataImpl queryMetaData = new QueryMetaDataImpl();
    queryMetaData.setQueryString("%");
    queryMetaData.setPageSize(10);
    queryMetaData.setCurrentPage(1);
    String engineInstName = "SES";
    String groupName = "runtime.EmpView";
    SearchGroup[] sgs = new SearchGroup[]
        {searchCtrl.getSearchGroup(engineInstName, groupName);
            queryMetaData.setSearchGroups(sgs);
            searchContext = ContextFactory.getSearchContext();
            searchContext.bindUser("scott");
            try
            {
                //clear tags so query tag through keywords
                queryMetaData.clearTags();
                queryMetaData.setQueryString("Black White");
                searchHits = searchCtrl.runQuery(searchContext, queryMetaData);
                //Zebra is found

                //filter by tags
                queryMetaData.setQueryString("%");
                queryMetaData.addTag("White");
                searchHits = searchCtrl.runQuery(searchContext, queryMetaData);
                //Mickey is found

                //filter by tags
                queryMetaData.addTag("White");
                queryMetaData.addTag("Black");
                searchHits = searchCtrl.runQuery(searchContext, queryMetaData);
                //Zebra is found

                //filter by tag that exists and tag that does not exist
                queryMetaData.clearTags();
                queryMetaData.addTag("Stripes");
                queryMetaData.addTag("Dots");
                searchHits = searchCtrl.runQuery(searchContext, queryMetaData);
                //no result is found
```
33.3.5 About Modifying Tags in Indexable Documents

You can override tags in the lifecycle methods provided by ECSF. For example, if required, you can add tags to an indexable document in the `IndexablePostProcess` by using the APIs provided on the `IndexableDocument`:

- `void addTags(String[] tags)` adds a list of strings as tags to the document. Duplicates are removed.
- `Collection<String> getTags()` returns a list of tags associated with the document.
- `void clearTags()` clears tags associated with the document.

33.3.6 About Registering Change Listeners

You can extend the incremental crawling mechanism by registering change listeners, which use the WebCenter Portal tagging framework to identify objects that need to be updated in the search engine. Implement the `oracle.ecsf.ChangeListener` interface in a search extension to customize the runtime logic that detects changes in the searchable objects. Example 33-3 illustrates a sample implementation of `ChangeListener`.

The method `Iterator getChangeList (SearchContext ctx, String changeType)` returns an iterator (`ChangeIterator`) over a list of primary keys for the searchable object (`ctx.getSearchableObject`).

The primary keys returned from the change listener are logged in the ECSF change log table before an incremental control feed is constructed.

**Example 33-3    Sample Implementation of ChangeListener**

```java
public class TestPlugin extends DefaultSearchPlugin implements ChangeListener {
    public Iterator getChangeList(SearchContext ctx, String changeType) {
        return new MyChangeIterator(ctx, changeType);
    }
}

public class MyChangeIterator extends ChangeIterator {
    private Date lastTimeCrawled;
    private int mCount;
    private int mIndex = 0;

    public MyChangeIterator (SearchContext ctx, String changeType) {
        super(ctx, changeType);
        Date lastTimeCrawled = ctx.getSearchableObject().getLastTimeCrawled();
        //if this value is null, ECSF is doing an initial crawl
        if(lastTimeCrawled==null)
33.4 About Enabling Search on Tree-Structure-based Source Systems

In addition to supporting the crawling and searching of relational-based objects through view objects or Java Database Connectivity (JDBC), ECSF supports the capability to crawl data in hierarchical tree-based data structures, or tree structures, and to identify items for indexing to enable full-text search. A tree structure is a common data structure, such as a file system on a computer hard disk, used to organize a large number of items. Oracle Business Intelligence is an example of a source system that is organized in a tree structure.

By default, ECSF supports database crawling using Oracle ADF technology. To support crawling data that is organized in tree structures, you can extend the abstract implementation of the CrawlableFactory oracle.ecsf.data.tree.AbstractTreeWalker. The extension converts the data stored at tree nodes into documents that Oracle SES receives through Really Simple Syndication (RSS) feeds and indexes.

Note:
Crawling relies on the integrator’s implementation that is based on their underlying data structures and accessibility of the items to be indexed.

Note:
ECSF currently does not provide an interface for converting information in source systems to indexable documents. It is assumed that data structures of an indexable item are proprietary to the source system, and the interface for converting an item pertaining such a structure to indexable document are the responsibility of the integrator.

A searchable object holds metadata about the source system. It can be either an Oracle ADF view object with ECSF annotation or a class that extends oracle.ecsf.meta.SearchableObject. For tree structure-based source systems,
the searchable object is not view object-based, so ECSF does not load the view object. Instead, it loads a Java class that extends and implements the searchable object. When requested with a ConfigFeed URL, ECSF identifies the searchable object that holds the search metadata required by ECSF. Non-view object-based searchable objects can be grouped into search categories. Figure 33-4 illustrates the data flow for search on tree structure-based source systems.

**Figure 33-4** Data Flow for Search Using ECSF Tree Crawler

The integration of ECSF with the source system allows an Oracle SES instance to crawl the source system data. The Source System DataNode, which is exposed to the ECSF tree crawler, formulates the data structure and pulls data from the source system server using web services. ECSF converts the tree nodes into documents that Oracle SES receives through RSS feeds and indexes. The source system implements a security service. When data from the source system is indexed in the Oracle SES, it is guarded against access by the security service.

A security extension is needed to implement source and document-level security. A searchable object has a method of getting a search extension instance based on the metadata. To associate a search extension with a searchable object, configure it from the Search navigation tab of the overview editor in JDeveloper. For more information, see About Making View Objects Searchable.

ECSF also extends its attachment implementation to enable the implementer to open the stream for data pulling attachments that are associated with a particular node. For more information, see About Enabling Search on Fusion File Attachments.
33.4.1 How to Enable Search on Tree-Structure-Based Source Systems

Enable search on tree structure-based source systems by doing the following:

1. Crawling tree structures
2. Integrating search functionality for tree structures
3. Implementing administration using ECSF interfaces

33.4.2 How to Crawl Tree Structures

ECSF offers Java classes that provide support for traversing tree-structure-based data sources and identifying items for indexing.

Before you begin, install the ECSF seed data records in the ECSF schema of the Oracle Fusion Applications database using Seed Data Framework (SDF). For information, see Initializing Oracle Fusion Application Data Using the Seed Data Loader.

To implement search for tree-structure-based data sources, complete the following tasks:

1. Create a searchable object by extending the oracle.ecsf.data.tree.SearchableTreeObject class.
2. Implement a crawlable tree node to extract document metadata.
3. Extend AbstractTreeWalker to traverse the tree.
4. Implement security.
5. Implement the attachments interface to stream the documents to Oracle SES.
6. Deploy and start the ECSF servlet.
7. Configure Oracle SES to crawl ECSF.

33.4.2.1 About Creating a Searchable Object

Create a searchable object by extending the oracle.ecsf.data.tree.SearchableTreeObject class, as shown in Example 33-4. When you create a searchable object for a tree object, you create a Java class, the SearchableTreeDirectory class, that is part of your implementation Java archive (JAR) file.

You must override the initializeConfig method to create search metadata dynamically for your searchable object, including adding a custom attribute, setting the security plug-in class name, and so on.

You must override getCrawlableFactoryName() to return the class name of your extension of AbstractTreeWalker.

Override getFileName if you want to persist the last crawled timestamp in a particular location.

For information, see About Extending AbstractTreeWalker.

Example 33-4 Sample Code for SearchableTreeDirectory Class

```java
package oracle.ecsf.test.tree;
import oracle.ecsf.data.tree.SearchableTreeObject;
```
public class SearchableTreeDirectory extends SearchableTreeObject {
    //override the security plug to be used
    public void initializeConfig() {
        setPlugInName(SecurityPlugin.class.getName());

        //add a custom attribute
        //see processNode method where you must set attribute value
        //for this attribute for the indexable document
        DocumentDefinition docdef = this.getDocument();
        FieldDefinitionImpl field = new FieldDefinitionImpl("CUSTOM_ATTR");
        field.setBinding("CUSTOM_ATTR");
        //this flag indicates that this attribute be stored in SES
        field.setStored(true);
        docdef.addField(field);
    }

    //override the crawlable factory to be used
    public String getCrawlableFactoryName() {
        return FileTreeCrawler.class.getName();
    }

    //override the method to get the location and the name of the last crawled
timestamp file
    //this is optional, if it is not overridden, the default location is the
    //temporary directory of the
    //system, that is, in unix/linux, it is "/tmp" and in Windows, it is "c:/temp";
    //the default file name is
    //".ecsf." concatenated with the object name, in this case
    "oracle.ecsf.test.tree.SearchableTreeDirectory"
    public String getFileName() {
        return ".\ecsf.oracle.ecsf.test.tree.SearchableTreeDirectory";
    }
}

33.4.2.2 About Implementing a Crawlable Tree Node

A crawlable tree node represents a node in your tree structure. When crawled, each
node on the tree structure is wrapped in this object. This node is created by the
extension of AbstractTreeWalker. For more details on each method, see the
Javadoc for ECSF.

Implement a crawlable tree node by extending the
oracle.ecsf.data.tree.CrawlableTreeNode class to create a tree node class
called CrawlableTreeNodeImpl, as shown in Example 33-5, for extracting
document metadata. The tree node class models the tree structure repository.

In the processNode method, you must extract any metadata information about the
document (for example, setTitle, setKeyword, setContent) and populate the
indexable document that is passed in. You can also add any custom attributes to the
indexable documents, such as the isLeaf method that determines whether a node is a
folder or a document. This method is used by the crawlable factory to determine how
to traverse the tree.

Example 33-5    Sample Code for CrawlableTreeNodeImpl Class

package oracle.ecsf.data.tree;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.Date;
import oracle.ecsf.IndexableDocument;
import oracle.ecsf.meta.PrimaryKey;
import oracle.ecsf.data.tree.CrawlableTreeNode;

public class CrawlableTreeNodeImpl extends CrawlableTreeNode {
    String[] filesToHandle = new String[] {"java", "xml"};
    private File file;
    // Creates a node with a fully qualified name.
    public CrawlableTreeNodeImpl(String name) {
        super(name);
        file = new File(name);
        this.setPath(file.getPath());
        this.setName(file.getName());
    }
    // Internal. Determine whether a file should
    // be indexed.
    private boolean handleFile(File file) {
        if (file == null)
            return false;
        if (file.isDirectory())
            return false;
        for (String ext : filesToHandle) {
            if (file.getName().endsWith(ext))
                return true;
        }
        return false;
    }
    // Test if a file must be indexed.
    public boolean isIndexable() {
        return handleFile(file);
    }
    /*
     * Gets the last modified date.
     */
    public Date getLastModified() {
        if (file == null) {
            return null;
        } else {
            return new Date(file.lastModified());
        }
    }

    public void processNode(IndexableDocument doc) {
        try {
            StringBuffer indexContent = new StringBuffer();
            FileReader fileReader = new FileReader(file);
            BufferedReader br = new BufferedReader(fileReader);
            String strd = br.readLine();
            while (strd != null) {
                indexContent.append("\n" + strd);
                strd = br.readLine();
            }
            if (file.getName().endsWith("xml")) {
                PrimaryKey keys = new PrimaryKey();
                keys.put("Name", file.getName());
                keys.put("Path", file.getPath());
                doc.addAttachment(new AttachmentImpl(keys));
            }
            br.close();
        }
    }
}
fileReader.close();
doc.setContent(indexContent.toString());
doc.setFieldValue("CUSTOM_ATTR", "This is custom attributes to be saved in SES");
doc.setAttributeValue("CUSTOM_ATTR", "This value is custom attribute value");
doc.overrideAccessURL("path=" + file.getPath() + ";name=" + file.getName());
} catch (Exception e) {
    //handle errors
}

protected void init()
{
    super.init();
    if (!isLeaf())
    {
        File[] files = file.listFiles();

        if (files != null)
        {
            for (int i = 0; i < files.length; i++)
            {
                addChildNode(new CrawlableTreeNodeImpl(files[i].getAbsolutePath()));
            }
        }
    }
}

private static boolean isLink(File file) {
    try {
        if (!file.exists()) {
            return true;
        } else {
            String cnnpath = file.getCanonicalPath();
            String abspath = file.getAbsolutePath();
            return !abspath.equals(cnnpath);
        }
    } catch (IOException ex) {
        System.err.println(ex);
        return true;
    }

    public boolean isLeaf() {
        return file == null || !file.isDirectory() || isLink(file);
    }

33.4.2.3 About Extending AbstractTreeWalker

To provide the methods that enables Oracle SES to traverse tree structures and index the content items, extend AbstractTreeWalker by creating the abstract tree crawler class, called FileTreeCrawler, as shown in Example 33-6. FileTreeCrawler is invoked as a factory to create a crawlable tree node as defined in About Implementing a Crawlable Tree Node. The tree walker works with CrawlableTreeNode to provide the generic framework for traversing a tree structure and indexing the content items in the repository. It also uses TreeSplitter to divide the repository into branches to enhance the crawling performance.

When Oracle SES crawls the searchable object, an instance of FileTreeCrawler is created by ECSF. It traverses the tree structure by calling the methods defined in the FileTreeCrawler class. It goes through two passes. First, it collects only the structure information, and based on that, it forms a control feed that contains all the
folders that need to be visited for collecting documents. The `isIndexable` method determines whether a particular node is indexed by Oracle SES. You can also use it to place filters to control the type of document to be indexed.

When creating the `FileTreeCrawler` class, you must implement two constructors. One takes no parameters, and the other takes a searchable object where you can perform configurations specific to your application, if required.

**Example 33-6 Sample Code for FileTreeCrawler Class**

```java
package oracle.ecsf.data.tree;

import oracle.ecsf.meta.SearchableObject;

public class FileTreeCrawler extends AbstractTreeWalker {
    //
    // The application determine the root path.
    //
    String root="/home/cbrown";
    /**
    * Constructs a CrawlableFactory from a searchable object.
    */
    public FileTreeCrawler()
    {
        super();
    }

    public FileTreeCrawler(SearchableObject searchableObject)
    {
        super(searchableObject);
    }

    // Creates root node
    //
    public CrawlableTreeNode createCrawlable() {
        return new CrawlableTreeNodeImpl(root);
    }

    // Creates a node for a given path
    // if you cannot construct the path.
    //
    public CrawlableTreeNode createCrawlable(String path) {
        return new CrawlableTreeNodeImpl(path);
    }

    // Tests if a node is crawlable
    protected boolean isIndexable(CrawlableTreeNode node) {
        CrawlableTreeNodeImpl fNode = (CrawlableTreeNodeImpl) node;
        return fNode.isIndexable();
    }
}
```

### 33.4.2.4 About Implementing Security

Security rules on the documents indexed by Oracle SES is controlled by access control lists (ACLs). This is achieved by creating a search plug-in that implements the `oracle.ecsf.Secure` interface for the searchable object. When you implement a security extension for the searchable object, it is used to serve as the authorization module for indexed content in Oracle SES.
ECSF is secured by a pluggable security service, which is called when users try to search the indexed content. By default, ECSF provides an implementation based on Oracle Platform Security for Java. It is mainly used for authenticating and authorizing users into the system. However, if you have a non-Oracle security provider, or you want to use your own security implementation, you must extend the ECSF security service. Example 33-7 illustrates the skeleton of a security extension.

The security extension is a Java class that implements a securable interface. There are five methods available. Example 33-8 illustrates the skeleton of such class.

Your search plug-in must be assigned to the searchable object, as shown in Example 33-4.

**Example 33-7  Sample of Security Service**

```java
package oracle.ecsf.data.tree;

import oracle.ecsf.SearchContext;
import oracle.ecsf.SecurityService;
import oracle.ecsf.util.SecurityServiceFactory;

public class SecurityServiceImpl implements SecurityService{

    public String[] listSupportedFormats() {
        return new String[]{"BIEE"};
    }

    public String authenticate(SearchContext ctx, String userName, String password, String format) {
        return userName;
    }

    public String isUserValid(SearchContext ctx, String userName, String format) {
        return userName;
    }

    public String[] getSecurityValues(SearchContext ctx, String userName, String attrName, String objectId) {
        //Get keys for an attribute.
        return new String[]{};
    }
}
```

**Example 33-8  Sample Security Extension**

```java
package oracle.ecsf.data.tree;

import oracle.ecsf.IndexableDocument;
import oracle.ecsf.SearchContext;
import oracle.ecsf.SearchSecurityException;
import oracle.ecsf.Securable;

public class SecurityPlugin implements Securable{

    public boolean isAclEnabled(SearchContext ctx) {
        return true;
    }
}
```
// The ACL for the document
public String[] getAcl(SearchContext ctx, IndexableDocument doc) {
    return new String[]{"cbrown"};
}

// The keys to the ACL for the document for ctx.getUserName()
public String[] getSecurityKeys(SearchContext ctx) {
    return new String[]{};
}

// The ACL for the document, hashed against an attribute
public String[] getSecureAttrAcl(SearchContext ctx, IndexableDocument doc, String attributeName) {
    return new String[]{};
}

// The keys to the ACL for the document for ctx.getUserName(), hashed against an attribute
public String[] getSecureAttrKeys(SearchContext ctx, String attributeName) {
    return new String[]{};
}

// Returns configuration parameters for the extension, not used often
public String[] getSecurableParams()
    throws SearchSecurityException {
    return new String[]{};
}

33.4.2.5 About Implementing the Attachments Interface

Implementing the attachments interface, shown in Example 33-9, allows you to index binary files such as Word documents, Excel spreadsheets, PDF files, and so on.

After you implement this class, you can add any number of attachments to an indexable document in the processNode method of CrawlableTreeNode. The attachment must contain enough information in its primary key for you to open the attachment when requested by the read method, where you simply use the information stored in the primary key to read the document and write to the output stream passed to you.

Example 33-9 Attachments Interface Implementation

package oracle.ecsf.data.tree;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.OutputStream;
import java.util.HashMap;
import java.util.Map;
import oracle.ecsf.Attachment;
import oracle.ecsf.SearchContext;
import oracle.ecsf.meta.PrimaryKey;

public class AttachmentImpl implements Attachment {
    PrimaryKey primaryKey;

    public AttachmentImpl() {
    }

    public AttachmentImpl(Map map) {
        primaryKey = new PrimaryKey();
        primaryKey.putAll(map);
    }

    public AttachmentImpl(InputStream in)
        throws IOException {
        BufferedReader reader = new BufferedReader(new FileReader(in));
        // Read the attachment data...
    }

    public AttachmentImpl(String name)
        throws IOException {
        // Read the attachment data from a URL...
    }

    // Additional methods...

    @Override
    public String[] getAcl(SearchContext ctx, IndexableDocument doc) {
        return new String[]{"cbrown"};
    }

    @Override
    public String[] getSecurityKeys(SearchContext ctx) {
        return new String[]{};
    }

    @Override
    public String[] getSecureAttrAcl(SearchContext ctx, IndexableDocument doc, String attributeName) {
        return new String[]{};
    }

    @Override
    public String[] getSecureAttrKeys(SearchContext ctx, String attributeName) {
        return new String[]{};
    }

    @Override
    public String[] getSecurableParams()
        throws SearchSecurityException {
        return new String[]{};
    }

    // Additional methods...
}
public String getType() {
    return "text/xml";
}

public void initialize(SearchContext ctx, Map paramMap, PrimaryKey keys) {
    primaryKey = new PrimaryKey();
    primaryKey.putAll(keys);
}

public void read(SearchContext ctx, OutputStream stream) {
    String path = (String)primaryKey.get("Path");
    File file = new File(path);
    try {
        FileReader fileReader = new FileReader(file);
        BufferedReader br = new BufferedReader(fileReader);
        String strd = br.readLine();
        while (strd != null) {
            stream.write(strd.getBytes());
            strd = br.readLine();
        }
    } catch (IOException e) {
        //Handle errors
    }

    //Contains configuration parameters needed to read the attachment
    //This map can be an empty one as shown in this example.
    public Map getParameters() {
        Map parameters = new HashMap();
        return parameters;
    }

    //Name value pairs need to identify a specific attachment.
    public PrimaryKey getPrimaryKey() {
        return primaryKey;
    }

}

33.4.2.6 Deploy and Start the ECSF Servlet

Before Oracle SES can crawl your file system, you need an Oracle WebLogic Server instance to which you can deploy the ECSF servlet. For example, you can use the Integrated WebLogic Server container.

To deploy and start the ECSF server, do the following:

1. Create a Java project in JDeveloper.
2. Add ecsf.jar to its class path.
3. Develop your extensions.
4. Edit web.xml to add the servlet mapping in Example 33-10.
5. To use a custom security service, you must add –
   Doracle.ecsf.security.service=classnameOfSecurityService.
   Otherwise, Oracle Platform Security for Java security service is used.
6. Open ecsf.jar, right-click oracle.ecsf.feed.SearchFeedServlet, and select Run. The ECSF servlet starts, and the system is ready to be crawled.
Example 33-10  SearchFeedServlet Mapping

<servlet>
  <servlet-name>SearchFeedServlet</servlet-name>
  <servlet-class>oracle.ecsf.feed.SearchFeedServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>SearchFeedServlet</servlet-name>
  <url-pattern>/searchfeedservlet/*</url-pattern>
</servlet-mapping>

33.4.2.7 Configure Oracle SES to Crawl ECSF

Oracle SES must be installed, then configured to crawl ECSF. Install Oracle SES 11.2.1, then perform the following steps to configure Oracle SES with the necessary information for crawling tree structure-based data sources and identifying items for indexing.

To configure Oracle SES, do the following:

1. In Oracle SES, create a data source of data source type, for example, Oracle Fusion.
   For General, complete the following:
   • Name: your data source name
     Oracle SES supports string values of up to 100 characters.
   • Configuration URL: http://yourhost:port/appname/pathname/searchableObjectName/ConfigFeed
   • Authentication Type: NATIVE
   • User ID: username
   • Password: password
   • Scratch Directory: /tmp or c:\tmp or empty
   • Maximum number of connection attempts: 3
   For Authentication, complete the following:
   • Authorization: ACLs Controlled by the Source
     • User ID: username
     • Password: password
     • Business Component: searchableObjectName
     • Display URL Prefix: prefix of url for Oracle SES UI

2. Create a data source group:
   a. From Search Tab/Source Group, click the Create button.
   b. Provide a name.
c. Select **Fusion Data Type** to filter the data source.

d. Shuttle the data source to the right column.

e. Click **Finish**.

3. Activate the identity extension:

   a. From Global Settings, select **Identity Management Setup**.

   b. Select **Oracle Fusion** from the list.

   c. Click **Activate**. If it is already activated, deactivate it, then reactivate it.


   e. Enter your user name and password.

4. If Oracle SES is needed, create a federated trusted entity:

   a. From Global Settings, select **Federation Trusted Entities**.

   b. For Entity Name, enter your user name.

   c. Select the **Use Identity Plug-in** checkbox for the authentication option.

5. (Optional) Include the Oracle SES client Java archive (JAR) in your class path, as shown in **Example 33-11**, to query Oracle SES through API.

**Example 33-11 Class Path with Oracle SES Client Java Archive**

```java
public SearchHits doSearch(String soName, String query, int pageSize, int page) {
    SearchHits searchHits = null;
    String searchGroupName = GROUP_NAME;
    long engineInstId = -1;
    SearchContext searchContext = ContextFactory.getSearchContext();

    QueryMetaData qmd = new QueryMetaDataImpl();
    qmd.setQueryString(query);
    qmd.setPageSize(pageSize);
    qmd.setCurrentPage(page);

    SearchGroup group = new SearchGroup(searchGroupName, searchGroupName, -1, null, SearchContext.LOCAL);
    SearchGroup[] groups = new SearchGroup[] { group ];
    qmd.setSearchGroups(groups);

    SearchableGroup sg = MetaDataManager.getSearchableGroup(-1, searchGroupName);
    sg.addSearchableObject(soName);

    SearchableObject so = MetaDataManager.getSearchableObject(soName);
    searchContext.setSearchableObject(so);

    SearchEngine engine = SearchEngineManager.getSearchEngine(engineInstId);
    try {
        Searcher searcher = engine.getSearcher(searchContext);
        searchHits = searcher.search(searchContext, qmd);
    } catch (Throwable e)
```
33.4.3 About Integrating Search Functionality for Tree Structures

In the ECSF architecture, search related artifacts can be stored in any persistent storage. The metadata manager obtains these artifacts via configuration store abstraction. For Oracle Fusion Applications, a database-based configuration store is developed that can load configuration from the ECSF tables in the Oracle Fusion Applications database.

If you decide not to use the ECSF database-based configuration store for the metadata, you can implement your own configuration store by implementing the oracle.ecsf.meta.Configuration interface.

The search related artifacts are loaded into memory through VOConfiguration during runtime to be used for crawling, query, and administration. If information is not stored in a database, as in the case of Oracle Business Intelligence that stores its information in a tree structure-based source system, ECSF provides a flexible way to load the runtime objects through an interface (Configuration) placed between MetaDataManager and your configuration storage so that runtime objects are not restricted to being loaded from the database. Figure 33-5 illustrates the runtime architecture that includes the Configuration interface, which provides an alternative mechanism for loading runtime objects.

![Runtime Architecture with Configuration Interface](image)

A system property determines which configuration, VOConfiguration or a custom configuration, to use during runtime. In Figure 33-5, the BIConfiguration class is an example of a custom configuration that extends the existing AbstractConfiguration class.

Integrate search functionality for tree structures by extending the AbstractConfiguration class and using your configuration class.

33.4.3.1 About Setting the Configuration

The MetaDataManager class determines which configuration to call based on how you set the oracle.ecsf.configuration.class system property. For example,

```java
System.setProperty("oracle.ecsf_configuration_class", "oracle.ecsf.meta.impl.BIConfiguration");
```
sets the property to use BIConfiguration. If this property is not set, or an implementation class does not exist for this property, MetaDataManager calls VOConfiguration by default.

33.4.3.2 About Using the Configuration Interface

The Configuration interface, shown in Example 33-12, contains the methods implemented by VOConfiguration or a custom configuration to load runtime objects.

The Configuration interface includes the getEngineParameters() method so you can get and set the search engine parameters needed for ECSF runtime in the absence of a database.

Example 33-12  Configuration Interface

```java
public interface Configuration {
    /**
     * Returns all search engine instances.
     * @return a list of search engine instances available.
     */
    public List<MetaEngineInstance> getEngineInstances();

    /**
     * Returns engine parameters in a hashmap for a given engine instance.
     * @param engineId the engine instance id
     * @return HashMap configuration parameter
     */
    public Map getEngineParameters(long engineId);

    /**
     * Returns a searchable group for a given search engine instance, by name.
     * @param engineId The identification of the engine instance.
     * @param name The name of the searchable group.
     * @return a searchable group. Null if not found.
     */
    public SearchableGroup getSearchableGroup(long engineId, String name);

    /**
     * Returns a searchable object for a given search engine instance by class name.
     * @param engineId The identification of the engine instance.
     * @param name The class name of the searchable object.
     * @return a searchable object, null if not found.
     */
    public SearchableObject getSearchableObject(long engineId, String name);

    /**
     * Returns a list of searchable groups for a given search engine instance.
     * @param engineId The indentification of the engine instance.
     * @return a list of searchable groups, empty if not found.
     */
    public List<SearchableGroup> getSearchableGroups(long engineId);

    /**
     * Requests a reload of the configuration. Implementation should reload the objects from persistent storage.
     */
    public void reload();
}
```
33.4.3.3 About Using the AbstractConfiguration Class

The AbstractConfiguration class implements the necessary functionalities common to all non-database uptakers regardless of where the runtime object information is stored. You must complete the implementation by using your own custom class that extends AbstractConfiguration. For information, see About Extending AbstractConfiguration.

Example 33-13 illustrates the implementation of AbstractConfiguration. In this implementation, getSearchableGroups() loads the available groups from the Oracle SES instance. These groups are treated as external groups, and therefore advanced search and facets are not supported in this scenario.

In this implementation, there is only one engine instance but following MetaDataManager convention, a list of engine instances is returned.

Example 33-13  Sample AbstractConfiguration Implementation

```java
public abstract class AbstractConfiguration implements Configuration {
    public AbstractConfiguration() {
        super();
        MetaEngineInstance engine = new MetaEngineInstance();
        engine.setId(-1L);
        engines.add(engine);
    }

    public List<MetaEngineInstance> getEngineInstances() {
        return engines;
    }

    public List<SearchableGroup> getSearchableGroups(long engineId) {
        if(groups == null) {
            groups = new ArrayList<SearchableGroup>();
            try {
                Map map = SESAdmin.getGroups(ContextFactory.getSearchContext(), engineId);
                for (Object key : map.keySet()) {
                    SearchableGroup sg = new SearchableGroup(key.toString());
                    sg.setDisplayName((String)map.get(key));
                    sg.setIsExternal(true);
                    sg.setEngineInstanceId(engineId);
                    groups.add(sg);
                }
            } catch (SearchException e) {
                e.printStackTrace();
            }
            return groups;
        }

        public SearchableGroup getSearchableGroup(long engineId, String s)
for (SearchableGroup group : getSearchableGroups(engineId))
{
    if (group.getName().equals(s))
    {
        return group;
    }
}
return null;

public SearchableObject getSearchableObject(long engineId, String s)
{
    SearchableObject so = loadSOFromClass(s);
    if (so != null)
    {
        so.setSearchEngineInstanceId(engineId);
    }
    return so;
}

protected SearchableObject loadSOFromClass(String className)
{
    try
    {
        if (className.equals(SearchableObject.class.getName()))
        {
            return null;
        }
        Class cls =
            Thread.currentThread().getContextClassLoader().loadClass(className);
        if (cls != null)
        {
            Object object = null;
            try
            {
                object = cls.newInstance();
            }
            catch (InstantiationException e)
            {
                return null;
            }
            catch (IllegalAccessException e)
            {
                return null;
            }
            if (object instanceof SearchableObject)
            {
                //this is the only path a searchable object will be created
                SearchableObject so = (SearchableObject)object;
                so.setDocument(new VODocumentImpl("root"));
                return so;
            }
        }
        else
        {
            return null;
        }
    }
}
### 33.4.3.4 About Implementing Searchable Object Classes

Since the searchable objects are not stored in the database, you must implement a Java class to define each searchable object. Example 33-14 shows the sample implementation for `EmpView.java`.

Each searchable object is loaded using a class of the same name.

**Example 33-14  Sample Searchable Object Class**

```java
package runtime;  //package where runtime objects reside
import java.util.logging.Logger;
import oracle.ecsf.meta.AbstractDocumentDefinition;
import oracle.ecsf.meta.DocumentDefinition;
import oracle.ecsf.meta.FieldDefinitionImpl;
import oracle.ecsf.meta.SearchableObject;
import oracle.ecsf.util.ECSFLoggerFactory;

public class EmpView extends SearchableObject
{
    private static Logger sLogger =
        ECSFLoggerFactory.getLogger(EmpView.class.getName());

    private static final String DATE_FORMAT = "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'";

    public EmpView()
    {
        super("*");
        setName(getClass().getName());
    }

    public EmpView(String name)
    {
        super(name);
    }

    /**
     * Override to initialize document.
     * @param document
     */
```
public final void setDocument(AbstractDocumentDefinition document) {
    super.setDocument(document);
    initDoc(document);
}

private void initDoc(AbstractDocumentDefinition document)
{
    FieldDefinitionImpl field = new FieldDefinitionImpl("Ename");
    field.setBinding("ENAME");
    field.setPrimaryKey(false);
    field.setStored(true);
    documentDefinition.addField(field);
    field = new FieldDefinitionImpl("Empno");
    field.setBinding("EMPNO");
    field.setPrimaryKey(true);
    field.setStored(true);
    documentDefinition.addField(field);
    field = new FieldDefinitionImpl(DocumentDefinition.ECSF_SO_NAME);
    field.setBinding(DocumentDefinition.ECSF_SO_NAME);
    field.setPrimaryKey(false);
    field.setStored(true);
    documentDefinition.addField(field);
}

33.4.3.5 About Extending AbstractConfiguration

You can extend the AbstractConfiguration class with the BIConfiguration class, as shown in Example 33-15, to define your own way of loading runtime objects that is specific to your environment.

Example 33-15 Sample Implementation for BIConfiguration

package oracle.ecsf.meta.impl;

import java.util.HashMap;
import java.util.Map;

public class BIConfiguration extends AbstractConfiguration {
    public BIConfiguration() {
    }
    public Map getEngineParameters(long engineId) {
        HashMap map = new HashMap();
        map.put("SES_ADMIN_USERNAME", "searchsys");
        map.put("SES_ADMIN_PASSWORD", "welcome1");
        map.put("SES_ADMIN_SESSION_TIMEOUT", "10");
        map.put("SES_ADMIN_SERVICE", "http:// sesserver.com:7777/search/api/admin/AdminService");
        map.put("SES_QUERY_PROXY_USERNAME", "scott");
        map.put("SES_QUERY_PROXY_PASSWORD", "tiger");
        map.put("SES_QUERY_SESSION_TIMEOUT", "10");
        map.put("ECSF_DATA_SERVICE", "http://wlsserver.com:7101/approot/searchfeedservlet/");
    }
}
map.put("ECSF_SECURITY_USERNAME", "scott");
map.put("ECSF_SECURITY_PASSWORD", "tiger");
map.put("ECSF_SECURITY_SERVICE",
"http://wlsserver.com:7101/approot/searchfeedservlet/");
map.put("ECSF_REDIRECT_SERVICE",
"http://wlsserver.com:7101/approot/searchfeedservlet/");

return map;
}

//add search group and search object used for SESAdmin
public List<SearchableGroup> getSearchableGroups(long engineId)
{
    if(searchGroups == null)
    {
        searchGroups = new ArrayList(super.getSearchableGroups(engineId));
        SearchableGroup sgAdmin = new SearchableGroup(GROUP_NAME_ADMIN);
        sgAdmin.setIsExternal(false);
        sgAdmin.setEngineInstanceId(ENGINE_ID);
        SearchableObject so = super.loadSOFromClass(OBJECT_NAME_ADMIN);
        sgAdmin.addSearchableObject(so);
        searchGroups.add(sgAdmin);
    }
    return searchGroups;
}

private List<SearchableGroup> searchGroups = null;

private static final long ENGINE_ID = -1L;
private static final String GROUP_NAME_ADMIN = "runtime.EmpViewAdminTest";
private static final String OBJECT_NAME_ADMIN = "runtime.EmpViewAdminTest";

### 33.5 About Managing Recent Searches

Recent Searches will save the user's top 10 most recent searches. The maximum number of recent searches that are saved can be configured. Recent Searches will display in the UI with the search keywords as their name and the user can use the recent search to requery. Recent Searches can be deleted one at a time, or all Recent Searches for a specified user can be deleted.

The Recent Search feature uses the SaveSearch database table and dataobjects, which already exist. The SEARCH_TYPE column added to the ECSF_SVSEARCH table specifies whether a search is SAVED or RECENT. If necessary, make sure to update the SavedSearchManager so that it handles this column.

Recent Searches uses these components:

- **RecentSearchManager Class**
  
The RecentSearchManager will manage the creation, deletion and retrieval of recent searches from the database.

- **SavedSearchManager Class**
  
  When a Saved Search is created, the SEARCH_TYPE column in the ECSF_SVSEARCH database table will be set to the appropriate value, and the SavedSearchManager will retrieve only searches that are of type SAVED.
33.5.1 About Using the RecentSearchManager API

This class, shown in Example 33-16, manages the Recent Searches.

Example 33-16  RecentSearchManager API

```java
/**
 * Returns a List of recent RecentSearch data objects for the
 * current user defined in the SearchContext
 * @param ctx  the current SearchContext
 * @return List< RecentSearch > list of recent RecentSearches
 * @throws SearchException
 */
public List<RecentSearch> getRecentSearches(SearchContext ctx)
    throws SearchException

/**
 * Returns a List of recent RecentSearch data objects for the
 * current user defined in the SearchContext and the specified
 * caller context
 * @param ctx - the current SearchContext
 * @param callerCtx - the caller context
 * @return List<RecentSearch> - list of recent RecentSearches ordered
 * from most recent to oldest
 * @throws SearchException
 */
public List<RecentSearch> getRecentSearches(SearchContext ctx, String callerCtx)
    throws SearchException

/**
 * Creates a new recent search
 * @param ctx - the SearchContext
 * @param searchDescription - the description of the saved search
 * @param queryDetails - recent saved search query information
 * @return RecentSearch
 * @throws SearchException
 */
public RecentSearch createRecentSearch(SearchContext ctx,
    String searchDescription,
    QueryMetaData queryDetails)
    throws SearchException

/**
 * Creates a new recent search
 * @param ctx - the SearchContext
 * @param searchDescription - the description of the saved search
 * @param queryDetails - recent saved search query information
 * @param callerCtx - the caller context
 * @return RecentSearch
 * @throws SearchException
 */
public RecentSearch createRecentSearch(SearchContext ctx,
    String searchDescription,
    QueryMetaData queryDetails,
### 33.5.2 Understanding How Recent Searches Are Processed

Recent Searches are managed only from the current application container, so the RecentSearchManager only handles recent searches that are found in the local application database, regardless of the ECSF Scope (LOCAL/GLOBAL). RecentSearchManager will not call to other search applications for recent searches.

**Important:** When a RecentSearch query is run, the query is run as usual using the defined Scope and Federation, so a federated query will go across the wire to the search applications if required.

Recent Searches are not implicitly created by ECSF. For a Recent Search to be saved, the `RecentSearchManager.createRecentSearch` API must be called.

The call to get the Recent Searches will return a list ordered from most recent to oldest recent search.

For performance, the list of recent searches returned will not contain the RecentSearchDetails information for the Recent Searches. The fact that this information is not available in the Recent Search dataobject is transparent to the client. When the client wants the details and calls `RecentSearch.getRecentSearchDetails()` on the Recent Search, the dataobject automatically will retrieve the RecentSearchDetails from the database.

By default, the maximum number of Recent Searches for each user is limited to 10. This number can be configured using a Java system property:
oracle.ecsf.recent.search.max.num

This property can be set using the Java -D option or the ecsf.properties file.

---

**Note:**

If the oracle.ecsf.recent.search.max.num property is set to an invalid number or 0, recent searches are disabled and a message is printed to the log.

The isEnabled(SearchContext ctx) API indicates whether Recent Search functionality is enabled. If Recent Search is disabled, calling the other methods in RecentSearchManager will result in an UnsupportedOperationException.

---

When Recent Search records are retrieved from the database, the SQL limits the number of records based on the system property (or the default of 10), so that only the most recent records are returned.

When the RecentSearchManager creates a new Recent Search, it first checks if the user already has the maximum number of Recent Searches stored in the database. If the user does not, a new Recent Search is created. If the user has exceeded the maximum number of Recent Searches allowed, the oldest Recent Search record is deleted and then the new Recent Search is saved into the database.

During the creation of a new Recent Search, at most one older Recent Search record may be deleted. Because the maximum number of Recent Searches is configurable, if this property is changed, there may be more Recent Search records in the database than allowed per user, and the Recent Search records should be manually cleaned from the database to clean up any extra records.

When a new Recent Search is created, due to the ECSF_SVSEARCH database table constraint on the NAME being unique per user, the RecentSearchManager will generate a unique name for the Recent Search using the database RowId for the new record and the queryString input by the user:

```java
String searchName = rowId + ": " + queryDetails.getQueryString();
```

This will maintain the constraint of having a unique name for each recent SavedSearch record. The UI will want to display the keyword string as the Recent Search name. The keyword string can be obtained from the SavedSearch dataobject using its getKeywordSrchStr method:

```java
    String recentSearchDisplayName = recentSearch.getKeywordSrchStr();
```

To run a Recent Search, the RecentSearchDetails can be retrieved from the RecentSearch object and then SearchCtrl.runQuery can be used to run the query on the queryMetaData in the RecentSearchDetails. If the query is federated, it will run as usual using federation. All of the necessary details for this are stored in the queryMetaData object:

```java
    RecentSearchDetails recentSearchDetails = recentSearch.getRecentSearchDetails();
    QueryMetaData qmd = recentSearchDetails.getQueryDetails();
    SearchHits hits = searchCtrl.runQuery(qmd);
```

The callerCtx passed into some of the RecentSearchManager methods is a column used by the UI to tag searches. For example, the Global Search UI will only want to display recent searches that were performed inside the Global Search UI, so the callerCtx will be used when creating and retrieving searches.
Database Schema for Recent Searches

Recent Search uses the ECSF_SVSEARCH and ECSF_SVSEARCH_DETAILS database tables. To support Recent Search, these schema changes have been made:

- SEARCH_TYPE has been added to ECSF_SVSEARCH:
  ```
  SEARCH_TYPE VARCHAR2(20) default 'SAVED' NOT NULL
  ```

- The unique constraint on the ECSF_SVSEARCH table includes the SEARCH_TYPE column.
  ```
  CONSTRAINT ECSF_SVSEARCH_UK1 UNIQUE (NAME, USERID, CALLER_CTX, SEARCH_TYPE)
  ```

- The existing ECSF_SVSEARCH_N1, ECSF_SVSEARCH_N2, and ECSF_SVSEARCH_U2 Indexes for the ECSF_SVSEARCH table include the SEARCH_TYPE column.
  ```
  CREATE INDEX ECSF_SVSEARCH_N1 ON ECSF_SVSEARCH (USERID, SEARCH_TYPE);
  CREATE INDEX ECSF_SVSEARCH_N2 ON ECSF_SVSEARCH (USERID, CALLER_CTX, SEARCH_TYPE);
  CREATE INDEX ECSF_SVSEARCH_U2 ON ECSF_SVSEARCH (NAME, USERID, CALLER_CTX, SEARCH_TYPE);
  ```

- The ECSF_SVSEARCH_DETAILS table is constrained on the Foreign Key to ECSF_SVSEARCH so that corresponding search details are automatically deleted whenever an ECSF_SVSEARCH record is deleted.
  ```
  ALTER TABLE ECSF_SVSEARCH_DETAILS ADD 
  ( CONSTRAINT ECSF_SVSEARCH_DETAILS_FK1 
    FOREIGN KEY (SVSEARCH_ID) REFERENCES ECSF_SVSEARCH (ID) 
    ON DELETE CASCADE 
  )
  ```

RecentSearch Dataobject

The RecentSearch and RecentSearchDetails dataobjects are used. A RecentSearch dataobject is just like SavedSearch except for the details method:

```java
/**
 * Retrieve the RecentSearchDetails for this recent search.
 * @return recent search details
 */
public RecentSearchDetails getRecentSearchDetails()
```

33.6 Set Up Federated Search

ECSF provides the services and federation to enable users to search across Oracle Fusion Applications product families or across multiple Oracle SES instances. For more information, see the "Managing Search with Oracle Enterprise Crawl and Search Framework" chapter in the Administrator's Guide.

Set up ECSF services and federation by:

1. Creating the SearchDB connection on Oracle WebLogic Server
2. Updating the application deployment profile with the Target Directory for Searchable Objects
3. Updating the application to point to the ECSF Service shared library
4. Adding the ECSF Runtime Library
5. Adding searchable objects and their dependencies to the Search application
6. Setting the system parameter for web service
7. Packaging and deploying the Search application
8. Setting up the ECSF client for federation
9. Setting the SearchContext scope to GLOBAL
10. Integrating federation across Oracle Fusion Applications product families

33.6.1 How to Create the SearchDB Connection on Oracle WebLogic Server Instance

The Oracle WebLogic Server instance to which the application is deployed must have a SearchDB connection. The application deployment descriptors are set so that the application does not automatically generate and synchronize weblogic-jdbc.xml descriptors during deployment. This setting prevents you from receiving the deployment error **No credential mapper entry found for password indirection** when you package or deploy from the command line or from Oracle JDeveloper. Because of this, you must manually create the SearchDB connection on Oracle WebLogic Server instance.

To create the SearchDB connection, do the following:

1. In the Domain Structure tree of the Oracle WebLogic Server Administration Console, navigate to Services, then JDBC, then Data Sources.
2. See if there is any data source with Java Naming and Directory Interface (JNDI) value jdbc/SearchDBDS. If not, proceed to the next step.
3. Click the New.
4. On the Connection Properties page, complete the fields with the following values:
   - **JNDI Name**: jdbc/SearchDBDS
   - **Database Type**: Oracle
   - **Database Driver**: Oracle Driver (Thin) 901,92,10,11
5. Click Next, and complete the configuration according to your data source.

33.6.2 How to Update the Application Deployment Profile with the Target Directory for Searchable Objects

All searchable objects and their dependencies must be packaged within the Search application enterprise archive (EAR) file for each product family. You must set the target directory for the Java archive (JAR) files containing the searchable objects and their dependencies in the application deployment descriptor so that they are packaged with the enterprise archive.

To set the target directory, do the following:

1. In the Application Navigator, right-click the application name and navigate to Application Properties, then Deployment.
The deployment descriptor file (for example, Search_Application1.ear, where Search is your application name) appears in the left pane.

2. Click the deployment descriptor file and select Edit.

3. From the Edit EAR Deployment Profile Properties menu navigate to File Groups > VOLib > Contributors, click Add, and set the directory to point to {FULL_PATH_TO_BUILD_FILE}/deploy/lib/, for example, /ade/view_name/fusionapps/crm/deploy/lib.

This directory is empty, but during the pre-enterprise archive step in the build file the directory becomes populated with all of the dependent Java archive (JAR) files. Including this directory in the application deployment descriptor ensures that these dependent Java archive (JAR) files are packaged with the enterprise archive.

4. Click OK.

33.6.3 About Updating the Application to Reference the ECSF Service Shared Library

The ECSF Service shared library eliminates the need for ECSF libraries to be packaged into the Search application for each product family. Instead, applications that depend on ECSF libraries can reference the ECSF shared library that is deployed to the Oracle WebLogic Server instance. The ECSF Service shared library contains the following Java archive (JAR) files:

- ecsf_MiddleTier.war
- ecsf_MiddleTier.jar
- ecsf_Common.jar

You must update the Oracle WebLogic Server deployment descriptor file (weblogic-application.xml) by adding the reference to the ECSF Service shared library (oracle.ecsf.service), as shown in Example 33-17.

Replace REPLACE _CONTEXT_ROOT with the context root that is desired for the Search application’s ECSF Service.

The ECSF Service shared library is automatically deployed to the Integrated WebLogic Server instance by the ECSF extension in JDeveloper through the JDeveloper Application Development Runtime Service (ADRS). However, you must manually deploy the ECSF Service shared library to the standalone Oracle WebLogic Server.

Example 33-17 Reference to the ECSF Service Shared Library

```xml
<library-ref>
  <library-name>oracle.ecsf.service</library-name>
</library-ref>

<library-context-root-override>
  <context-root>searchservice</context-root>
  <override-value>REPLACE _CONTEXT_ROOT</override-value>
</library-context-root-override>
```

33.6.4 How to Add the ECSF Runtime Library

After you update the application deployment profile, update the project by adding the ECSF Runtime Library.

To add the Runtime Library, do the following:
1. Right-click the project and select **Project Properties**.

2. Select the **Libraries and Classpath** category and click the **Add Library** button.

3. In the Add Library dialog, select **ECSF Runtime Server** from the list of available libraries.

4. Click **OK** to save your selection and close the Add Library dialog.

### 33.6.5 About Setting the System Parameter for Web Service

Set the `oracle.ecsf.service.ws.timeout` system parameter to specify the web service timeout value in milliseconds. If no value is specified, then 90,000 milliseconds is used. You can set the system parameter in either the Java system properties or in the `ecsf.properties` file.

#### 33.6.5.1 Set the System Parameter in Java System Properties

Set the `oracle.ecsf.service.ws.timeout` system parameter in Java System Properties to specify the web service timeout value.

To set the system parameter, do the following:

1. In the **Application Navigator**, right-click **ViewController** and select **Project Properties**.

2. In the Project Properties dialog, select **Run/Debug/Profile** in the left panel.

3. Select **Default** in the Run Configurations list, then click the **Edit** button.

4. In the Edit Run Configuration dialog, select **Launch Settings** in the left panel.

5. Enter `-Doracle.ecsf.service.ws.timeout=n` (where `n` is the desired value in milliseconds) in the **Java Options** field, then click **OK**.

6. Click **OK**.

#### 33.6.5.2 About Setting the System Parameter in the `ecsf.properties` File

Set the `oracle.ecsf.service.ws.timeout` system parameter in the `ecsf.properties` file to specify the web service timeout value by adding the following line to the `ecsf.properties` file available in the application classpath:

```
oracle.ecsf.service.ws.timeout=n
```

where `n` is the desired value in milliseconds.

### 33.6.6 About Packaging and Deploying the Search Application

When the application deployment descriptor points to the right directory you can run the **ant** targets to package and deploy the EAR file. You can run the **ant** targets from the command line or from Oracle JDeveloper.

#### 33.6.6.1 About Running the **ant** Targets from the Command Line

Package and deploy the Search application by issuing the following commands in the directory where the build file is located:

```
ant -f build-crmsearch.xml ear
ant -Ddeployenvfile=/scratch/deploy.xml -f build-crmsearch.xml deploy
```
where `build-crmsearch.xml` is the build file for the Search application.

The enterprise archive step in the build file includes the pre-enterprise archive step, so there is no need to manually run the pre-enterprise archive step. The enterprise archive target also runs a post-enterprise archive step that deletes all the files from the `deploy/lib` directory after the enterprise archive is packaged, resulting in a clean folder.

### 33.6.6.2 Run the ant Targets from Oracle JDeveloper

You can use Oracle JDeveloper to package and deploy the Search application.

To package and deploy the Search application, do the following:

**To package and deploy the Search application from Oracle JDeveloper:**

1. Run the pre-enterprise archive steps by opening the build file, right-clicking the open file editor, and selecting `Run ant target > lrg - writeGraph > pre-ear`.

   This step runs the pre-enterprise archive steps that are required for copying required Java archive (JAR) files to the directory specified in the Search application deployment descriptor. All the files in that directory is packaged into the enterprise archive file in the `APP-INF/lib` file where the ECSF WAR can locate them.

2. Package the enterprise archive by right-clicking the application and selecting `Deploy > To ear`.

3. Deploy the enterprise archive by right-clicking the application and selecting `Deploy > To IntegratedWebLogicServer`.

### 33.6.7 About Updating the Search Application with New Searchable Objects or Dependencies

The Search application must be updated if there are new searchable objects to add to the application or if any of the dependencies for existing searchable objects change.

If there are no new searchable objects but dependencies have changed, you only need to run the `dependentJar` ant target and package and deploy the enterprise archive (for information, see About Packaging and Deploying the Search Application).

However, if you are adding new searchable objects to the application, then you must add the new searchable objects to the Search application build file and run the `dependentJar` ant target, then repackage and redeploy the enterprise archive (for information, see About Packaging and Deploying the Search Application).

### 33.6.8 How to Set Up the ECSF Client Application for Federation

To connect to the Search application for each Oracle Fusion Applications product family (collectively called global search applications), you must configure the client application so that when it sends the Search application server a request, it also sends valid encrypted proxy user credentials to the server.

The client that calls the Search application must be configured with information on where to find the global Search applications. This information is stored in the `connections.xml` of the client application. The `connections.xml` file of the client application must contain a reference name element corresponding to each remote ECSF component to which the client application connects.

Set up the ECSF client application by:
1. Adding encryption keys to `cwallet.sso` and `default-keystore.jks`
2. Adding the keystore to `jps-config.xml`
3. Creating the proxy user
4. Updating `connections.xml`

### 33.6.8.1 About Adding Encryption Keys to `cwallet.sso` and `default-keystore.jks`

The security header is encrypted before being sent to the server, so the `cwallet` and `default-keystore` files for both the client and server must be configured for the encryption to function properly. Use the Oracle WebLogic Scripting Tool to create the encryption keys in `cwallet.sso` and `default-keystore.jks`.

The following four new entries appear in `cwallet.sso` and `default-keystore.jks`:

```bash
> createCred(map="oracle.wsm.security", key="keystore-csf-key", user="owsm", password="welcome1", desc="Keystore key")
> createCred(map="oracle.wsm.security", key="enc-csf-key", user="orakey", password="welcome1", desc="Encryption key")
> createCred(map="oracle.wsm.security", key="sign-csf-key", user="orakey", password="welcome1", desc="Signing key")
> createCred(map="oracle.wsm.security", key="basic.credentials", user="weblogic", password="weblogic1", desc="User credentials key")
```

Oracle WebLogic Scripting Tool updates the files directly on the server you specify, so no redeployment is necessary.

### 33.6.8.2 About Adding the Keystore to `jps-config.xml`

In order for the client to be able to use the encryption keystore entries, you must configure the `jps-config.xml` file for the client application.

Add the following entries to `jps-config.xml`:

- **Under serviceProviders:**
  ```xml
  <serviceProvider type="KEY_STORE" name="keystore.provider"
  class="oracle.security.jps.internal.keystore.KeyStoreProvider">
  </serviceProvider>
  ```

- **Under serviceInstances:**
  ```xml
  <serviceInstance name="keystore" provider="keystore.provider" location="/.
default-keystore.jks">
  <description>Default JPS Keystore Service</description>
  <property name="keystore.type" value="JKS"/>
  <property name="keystore.csf.map" value="oracle.wsm.security"/>
  <property name="keystore.pass.csf.key" value="keystore-csf-key"/>
  <property name="keystore.sig.csf.key" value="sign-csf-key"/>
  <property name="keystore.enc.csf.key" value="enc-csf-key"/>
  </serviceInstance>
  ```

- **Under the default jpsContext:**
  ```xml
  <serviceInstanceRef ref="keystore"/>
  ```
33.6.8.3 About Creating the Proxy User

The proxy user must exist on both the client and server. The client's cwallet.sso must also include an entry for the proxy user so that the username and password can be encrypted when they are placed in the security header before being sent to the server. Use Oracle WebLogic Scripting Tool to create an entry for the user in cwallet.sso, as shown in Example 33-18.

The key (in this example, test.user) for the new entry is used in connections.xml.

Example 33-18  Sample Proxy User Entry

createCred(map="oracle.wsm.security", key="test.user", user="weblogic", password="weblogic1", desc="User credentials key")

33.6.8.4 Update connections.xml

The connection between the ECSF client and each of the remote ECSF service components is defined in the connections.xml file in the ECSF client. The connections.xml file contains a list of reference name elements that correspond to each ECSF service component. You must edit the connections.xml file to define the application server connection parameters.

To define the connection parameters, do the following:

1. Expand Application Resources, then Descriptors, then ADF Meta-INF, and open the connections.xml file.
2. Add the reference name elements, as shown in Example 33-19.

Web service security must be enforced by policy at the domain or instance level by configuration.
3. Save.
4. Redeploy the client application.

Example 33-19  Sample Reference Element

<Reference name="\\oracle/ecsf/service/query/common/crm/\}SearchService" className="oracle.adf.model.connection.webservice.impl.WebServiceConnectionImpl" xmlns="">
  <Factory className="oracle.adf.model.connection.webservice.api.WebServiceConnectionFactory"/>
  <RefAddresses>
    <XmlRefAddr addrType="WebServiceConnection">
      <Contents>
          <model name="\\oracle/ecsf/service/query/common\} AppModuleSearchService" xmlns="http://example.com/ws/model">
            <service name="\\oracle/ecsf/service/query/common\} AppModuleSearchService">
              <port name="AppModuleSearchServiceSoapHttpPort" binding="/oracle/ecsf/service/query/common\} AppModuleSearchServiceSoapHttp">
              </port>
            </service>
          </model>
        </wsconnection>
      </Contents>
    </XmlRefAddr>
  </RefAddresses>
</Reference>
33.6.9 About Setting the SearchContext Scope to GLOBAL

ECSF Query APIs can either invoke the ECSF service component internally (GLOBAL) or locally (LOCAL). To set the ECSF calls to be routed to the ECSF service component, you must set the scope of the search context to GLOBAL, as shown in Example 33-20.

When the SearchContext scope is set to GLOBAL, the parameters defined for the remote engine instance in the database are used to access metadata objects and perform query related functions on the remote engine instance. For more information, see the "Managing Search with Oracle Enterprise Crawl and Search Framework" chapter in the Administrator's Guide.
Example 33-20  Client Methods

SearchContext ctx = ContextFactory.getSearchContext();
ctx.setScope("GLOBAL");
ArrayList<SearchEngineInstance> engineInstances =
(ArrayList<SearchEngineInstance>)searchCtrl.getEngineInstances();

33.6.10 About Integrating Federation Across Oracle Fusion Applications Product Families

Use the ECSF API, as shown in Example 33-21, to integrate federation across Oracle Fusion Applications product families.

In the example, ECSF runtime gets the runtime.EmpView search category (search group) from an engine instance.

Example 33-21  Sample API for Implementing Federated Search

SearchCtrl searchCtrl = new SearchCtrl();
SearchHits searchHits = null;
QueryMetaDataImpl queryMetaData = new QueryMetaDataImpl();
queryMetaData.setQueryString("**");
queryMetaData.setPageSize(10);
queryMetaData.setCurrentPage(1);
ctx.setScope(SearchContext.GLOBAL);
ctx.setCurrLocale(Locale.US);

ArrayList<SearchGroup> allGroups = new ArrayList<SearchGroup>();
ArrayList<SearchGroup> searchGroups = new ArrayList<SearchGroup>();
ArrayList<SearchEngineInstance> engineInstances =
(ArrayList<SearchEngineInstance>)searchCtrl.getEngineInstances();
for (SearchEngineInstance engineInstance : engineInstances)
{
    allGroups.addAll(engineInstance.getSearchGroups());
}
for (SearchGroup searchGroup : allGroups)
{
    if (searchGroup.getName().equals("runtime.EmpView")
    || searchGroup.getName().equals("Service Request"))
    {
        searchGroups.add(searchGroup);
    }
}
SearchGroup sgs[] = searchGroups.toArray(new SearchGroup[searchGroups.size()]);
queryMetaData.setSearchGroups(sgs);
try
{
    searchHits = searchCtrl.runQuery(ctx, queryMetaData);
}
catch (Exception e)
{
    e.printStackTrace();
}

33.7 About Federating Oracle SES Instances

ECSF supports federation across Oracle SES instances, which allows users to query across multiple Oracle SES instances defined in the client’s own ECSF component. In Figure 33-6, the ECSF client is connected to one ECSF service component, which depends on one database and two Oracle SES instances.
Federation occurs on the client through the Searcher class. For each Oracle SES instance, ECSF runtime groups the search categories belonging to an Oracle SES instance and creates a federation node for it. Separate queries are issued in separate threads for each Oracle SES instance. The results from these queries are merged by ECSF runtime and returned to the user.

**Note:**

You cannot federate both Oracle SES instances and ECSF service components in the same query. ECSF runtime can only create federation nodes for either each Oracle SES instance (for federated Oracle SES) or each ECSF service component (for federated search).

**Example 33-22** illustrates how you can integrate federation across Oracle SES instances.

In the example, two Oracle SES search engine instances are defined.

**Example 33-22  Sample API for Integrating Federated Oracle SES**

```java
SearchContext ctx = ContextFactory.getSearchContext();
ctx.setScope(SearchContext.LOCAL);
SearchGroup[] sgs =
    new SearchGroup[] { new SearchGroup("runtime.EmpView", "runtime.EmpView", 1),
        new SearchGroup("runtime.EmpView", "runtime.EmpView", 17) };

queryMetaData.setSearchGroups(sgs);
try {
    searchHits = searchCtrl.runQuery(ctx, queryMetaData);
} catch (Exception e) {
    bException = true;
}
```
33.8 About Raising Change Events Synchronously

You can use a Java API to raise change events synchronously with `asynch` flag = `false` when searchable object records are modified. Raising events using SQL writes records to the `ECSF_SEARCHABLE_CHANGE_LOG` table of the database.

You can raise change events synchronously by implementing code like the sample code in Example 33-23.

You can use the following `ChangeType` parameters:

- `IndexableDocument.DELETE`
- `IndexableDocument.UPDATE`
- `IndexableDocument.INSERT`

When using the `SearchEventInvocation.raiseEvent(PrimaryKey eventPK, boolean useFabric)` method to create change log records, the keys in the `eventPK` map must be aliases of view object attributes. A view object’s alias can be found by opening the view object in JDeveloper, or by calling the `getBinding()` method of the appropriate `oracle.ecsf.meta.FieldDefinition` instance. Also, the `eventPK` map must contain an entry for every attribute that is part the primary key makeup. Otherwise, the `raiseEvent` method throws an exception.

For more information, see the Javadoc for ECSF.

**Example 33-23   Sample Code for Raising Events Using SQL**

```java
SearchEventInvocation searchEventInvocation = new SearchEventInvocation();
PrimaryKey primaryKey = new PrimaryKey();
primaryKey.put("PartyId", "12322");
SearchChangeLogEvent changeEvent = new SearchChangeLogEvent(pk);
changeEvent.setSearchObjectName("oracle.ecsf.search.demo.EmpVO");
changeEvent.setChangeType(IndexableDocument.INSERT);
searchEventInvocation.raiseEvent(changeEvent, false);
```

33.9 About Using the External ECSF Web Service for Integration

In addition to Java APIs, Oracle Enterprise Crawl and Search Framework (ECSF) provides an external web service for you to integrate ECSF with Oracle Fusion Applications. This web service allows you to build a custom search user interface that enables Oracle Fusion Applications users to search across multiple ECSF service components through a web service client. As an alternative to using Java APIs, you can invoke the external ECSF web service from Oracle Fusion Applications to perform query related functions in both LOCAL and GLOBAL scopes. Using a web service client, users can query across multiple Oracle SES instances in LOCAL scope or across multiple ECSF service components in GLOBAL scope.

33.9.1 About Web Service Methods

The external ECSF web service reuses the web service already provided by an ECSF component and exposes the methods described in Table 33-1.
Table 33-1  ECSF Web Service Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSavedSearch(String userName, String savedSearchRequest)</td>
<td>Returns a saved search based on the name passed in to the savedSearchRequest parameter.</td>
</tr>
<tr>
<td>getSavedSearches(String userName, String savedSearchRequest)</td>
<td>Returns the saved searches based on the caller context passed in to the savedSearchRequest parameter.</td>
</tr>
<tr>
<td>saveSearch(String userName, String savedSearchRequest)</td>
<td>Saves the search passed in to the savedSearchRequest parameter.</td>
</tr>
<tr>
<td>deleteSearch(String userName, String savedSearchRequest)</td>
<td>Deletes the saved search passed in to the savedSearchRequest parameter.</td>
</tr>
<tr>
<td>getSavedSearchDetails(String userName, String savedSearchRequest)</td>
<td>Returns the saved search details based on the saved search passed in to the savedSearchRequest parameter.</td>
</tr>
<tr>
<td>search(String userName, String queryMetaDataRequest)</td>
<td>Returns search hits based on the request passed in to the QueryMetaData parameter.</td>
</tr>
<tr>
<td>getEngineInstances(String userName, String engineInstanceRequest)</td>
<td>Returns the engine instances based on the engine type ID passed in to the engineInstanceRequest parameter.</td>
</tr>
</tbody>
</table>

33.9.2 About ECSF Web Service WSDL and XSD

The Web Service Description Language (WSDL), shown in Example 33-24, defines the message endpoints and the request and reply messages of the ECSF web service. The XSD, shown in Example 33-25, defines the XML schema of the ECSF web service. Refer to the WSDL and XSD to understand and interact with the ECSF web service.

Example 33-24  ECSF Web Service WSDL

```xml
<wSDL:definitions
    name="AppModuleSearchService"
    targetNamespace="/oracle/ecsf/service/query/common/"
    xmlns:errors="http://xmlns.example.com/adf/svc/errors/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/
    xmlns:tns="/oracle/ecsf/service/query/common/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
    xmlns:types="/oracle/ecsf/service/query/common/types/">
    <wsdl:import namespace="http://xmlns.example.com/adf/svc/errors/"
        location="classpath:/META-INF/wsdl/ServiceException.wsdl"/>
    <wsdl:types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema">
            <import namespace="/oracle/ecsf/service/query/common/types/" schemaLocation="AppModuleSearchService.xsd"/>
        </schema>
    </wsdl:types>
    <wsdl:message name=" AppModuleSearchService_search">
        <wsdl:part name="parameters" element="types:search"/>
    </wsdl:message>
    <wsdl:message name=" AppModuleSearchService_searchResponse">
        <wsdl:part name="parameters" element="types:searchResponse"/>
    </wsdl:message>
</wSDL:definitions>
```
<wsdl:message name="AppModuleSearchService_saveSearch">
    <wsdl:part name="parameters" element="types:saveSearch"/>
</wsdl:message>

<wsdl:message name="AppModuleSearchService_saveSearchResponse">
    <wsdl:part name="parameters" element="types:saveSearchResponse"/>
</wsdl:message>

<wsdl:message name="AppModuleSearchService_getSavedSearches">
    <wsdl:part name="parameters" element="types:getSavedSearches"/>
</wsdl:message>

<wsdl:message name="AppModuleSearchService_getSavedSearchesResponse">
    <wsdl:part name="parameters" element="types:getSavedSearchesResponse"/>
</wsdl:message>

<wsdl:message name="AppModuleSearchService_getSavedSearchDetails">
    <wsdl:part name="parameters" element="types:getSavedSearchDetails"/>
</wsdl:message>

<wsdl:message name="AppModuleSearchService_getSavedSearchDetailsResponse">
    <wsdl:part name="parameters" element="types:getSavedSearchDetailsResponse"/>
</wsdl:message>

<wsdl:message name="AppModuleSearchService_getSavedSearch">
    <wsdl:part name="parameters" element="types:getSavedSearch"/>
</wsdl:message>

<wsdl:message name="AppModuleSearchService_getSavedSearchResponse">
    <wsdl:part name="parameters" element="types:getSavedSearchResponse"/>
</wsdl:message>

<wsdl:message name="AppModuleSearchService_getEngineInstances">
    <wsdl:part name="parameters" element="types:getEngineInstances"/>
</wsdl:message>

<wsdl:message name="AppModuleSearchService_getEngineInstancesResponse">
    <wsdl:part name="parameters" element="types:getEngineInstancesResponse"/>
</wsdl:message>

<wsdl:message name="AppModuleSearchService_getDisplayName">
    <wsdl:part name="parameters" element="types:getDisplayName"/>
</wsdl:message>

<wsdl:message name="AppModuleSearchService_getDisplayNameResponse">
    <wsdl:part name="parameters" element="types:getDisplayNameResponse"/>
</wsdl:message>

<wsdl:message name="AppModuleSearchService_deleteSearch">
    <wsdl:part name="parameters" element="types:deleteSearch"/>
</wsdl:message>

<wsdl:message name="AppModuleSearchService_deleteSearchResponse">
    <wsdl:part name="parameters" element="types:deleteSearchResponse"/>
</wsdl:message>

<wsdl:portType name="AppModuleSearchService">
    <wsdl:documentation/>
    <wsdl:operation name="search">
        <wsdl:input message="tns:AppModuleSearchService_search"/>
        <wsdl:output message="tns:AppModuleSearchService_searchResponse"/>
        <wsdl:fault name="ServiceException" message="errors:ServiceException"/>
    </wsdl:operation>
    <wsdl:operation name="saveSearch">
        <wsdl:input message="tns:AppModuleSearchService_saveSearch"/>
        <wsdl:output message="tns:AppModuleSearchService_saveSearchResponse"/>
        <wsdl:fault name="ServiceException" message="errors:ServiceException"/>
    </wsdl:operation>
    <wsdl:operation name="getSavedSearches">
        <wsdl:input message="tns:AppModuleSearchService_getSavedSearches"/>
        <wsdl:output message="tns:AppModuleSearchService_getSavedSearchesResponse"/>
        <wsdl:fault name="ServiceException" message="errors:ServiceException"/>
    </wsdl:operation>
    <wsdl:operation name="getSavedSearchDetails">
        <wsdl:input message="tns:AppModuleSearchService_getSavedSearchDetails"/>
        <wsdl:output message="tns:AppModuleSearchService_getSavedSearchDetailsResponse"/>
        <wsdl:fault name="ServiceException" message="errors:ServiceException"/>
    </wsdl:operation>
</wsdl:portType>
<wsdl:operation name="getSavedSearches">
<soap:operation soapAction="/oracle/ecsf/service/query/common/getSavedSearches"/></soap:operation>
</wsdl:operation>
About Using the External ECSF Web Service for Integration

Advanced Topics for ECSF 33-45
Example 33-25  ECSF Web Service XSD

```xml
    <import namespace="http://xmlns.example.com/adf/svc/errors/" schemaLocation="classpath:/META-INF/wsdl/ServiceException.xsd"/>

    <element name="search">
        <complexType>
            <sequence>
                <element name="userName" type="string"/>
                <element name="queryMetadataRequest" type="string"/>
            </sequence>
        </complexType>
    </element>

    <element name="searchResponse">
        <complexType>
            <sequence>
                <element name="result" type="string"/>
            </sequence>
        </complexType>
    </element>

    <element name="saveSearch">
        <complexType>
            <sequence>
                <element name="userName" type="string"/>
                <element name="savedSearchRequest" type="string"/>
            </sequence>
        </complexType>
    </element>

    <element name="saveSearchResponse">
        <complexType>
            <sequence>
                <element name="result" type="string"/>
            </sequence>
        </complexType>
    </element>
</schema>
```
About Using the External ECSF Web Service for Integration

<element name="getSavedSearches">
  <complexType>
    <sequence>
      <element name="userName" type="string"/>
      <element name="savedSearchRequest" type="string"/>
    </sequence>
  </complexType>
</element>

<element name="getSavedSearchesResponse">
  <complexType>
    <sequence>
      <element name="result" type="string"/>
    </sequence>
  </complexType>
</element>

<element name="getSavedSearchDetails">
  <complexType>
    <sequence>
      <element name="userName" type="string"/>
      <element name="savedSearchRequest" type="string"/>
    </sequence>
  </complexType>
</element>

<element name="getSavedSearchDetailsResponse">
  <complexType>
    <sequence>
      <element name="result" type="string"/>
    </sequence>
  </complexType>
</element>

<element name="getSavedSearch">
  <complexType>
    <sequence>
      <element name="userName" type="string"/>
      <element name="savedSearchRequest" type="string"/>
    </sequence>
  </complexType>
</element>

<element name="getSavedSearchResponse">
  <complexType>
    <sequence>
      <element name="result" type="string"/>
    </sequence>
  </complexType>
</element>

<element name="getEngineInstances">
  <complexType>
    <sequence>
      <element name="userName" type="string"/>
      <element name="engineInstanceRequest" type="string"/>
    </sequence>
  </complexType>
</element>

<element name="getEngineInstancesResponse">
  <complexType>
    <sequence>
      <element name="result" type="string"/>
    </sequence>
  </complexType>
</element>
33.9.3 About Web Service Request XSDs and XMLs

Each of the web service methods takes in a username and a request XML. The username is used to bind to the SearchContext. The request XML is passed to the SearchService to perform query related functions.

The request XMLs for the web service methods are based on the following request XSDs:

- SavedSearch Request XSD
- QueryMetaData Request XSD
- engineInstanceRequest Request XSD

33.9.3.1 SavedSearch Request XSD

The request XMLs for the `getSavedSearch()`, `getSavedSearches()`, `saveSearch()`, `deleteSearch()`, and `getSavedSearchDetails()` methods are based on the SavedSearch request XSD, shown in Example 33-26.

The SavedSearch request XSD describes the set of rules that the request XMLs must follow to be valid. Examples of valid request XMLs include the following:

- `getSavedSearch()`

```
<request>
  <name>ECSF_JUNIT_SVSEARCH</name>
  <callerctx>null</callerctx>
```
• getSavedSearches()

  <request>
  <callerctx>%</callerctx>
  <locale>en_us</locale>
  <scope>LOCAL</scope> /*use GLOBAL for global scope*/
  </request>

• saveSearch()

  <request>
  <savedSearch name="ECSF_JUNIT_SVSEARCH" id="1" compid="2">
    <description><![CDATA[Updated Description]]></description>
    <callerctx></callerctx>
    <query></query>
    <ispublic><![CDATA[false]]></ispublic>
    <userid><![CDATA[junit]]></userid>
    <detailsid><![CDATA[1]]></detailsid>
    <queryMetaData>
      <query><![CDATA[%]]></query>
      <page>1</page>
      <lang>en</lang>
      <pageSize>10</pageSize>
      <categories>
        <category name="runtime.EmpView" eid="1" compid="0"></category>
      </categories>
    </queryMetaData>
  </savedSearch>
  </request>

• deleteSearch()

  <request>
  <savedSearch name="DeleteSavedSearch" id="100010033142848" compid="2">
    <description><![CDATA[Junit Saved Search]]></description>
    <callerctx><![CDATA[CRM]]></callerctx>
  </savedSearch>
  </request>
• getSavedSearchDetails()

Example 33-26  SavedSearch Request XSD

<?xml version="1.0" encoding="US-ASCII" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
33.9.3.2 QueryMetaData Request XSD

The request XML for the `search(String userName, String queryMetaDataRequest)` method is based on the QueryMetaData request XSD, shown in Example 33-27.

The QueryMetaData request XSD describes the set of rules that the request XMLs must follow to be valid. An example of a valid request XML for `search()` is:

```xml
<request>
  <queryMetaData>
    <query><![CDATA[*]]></query>
    <page>1</page>
    <lang>en</lang>
    <pageSize>10</pageSize>
    <categories>
      <category name="runtime.EmpView" eid="1" compid="2"></category>
    </categories>
    <locale>en_us</locale>
    <scope>LOCAL</scope> /*use GLOBAL for global scope*/
  </queryMetaData>
</request>
```
Example 33-27   QueryMetaData Request XSD

<?xml version="1.0" encoding="US-ASCII" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.example.com"
  targetNamespace="http://www.example.com"
  elementFormDefault="qualified">
  <xsd:element name="request">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="queryMetaData"/>
        <xsd:element name="locale" type="xsd:string"/>
        <xsd:element name="scope" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="queryMetaData">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="query" type="xsd:string"/>
        <xsd:element name="page" type="xsd:integer"/>
        <xsd:element name="lang" type="xsd:string"/>
        <xsd:element name="pageSize" type="xsd:integer"/>
        <xsd:element name="searchCtrl" type="xsd:string" minOccurs="0"/>
        <xsd:element name="soname" type="xsd:string" minOccurs="0"/>
        <xsd:element name="facetPaths" minOccurs="0">  
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="facetPath" maxOccurs="unbounded">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="value" maxOccurs="unbounded" minOccurs="0">  
                      <xsd:complexType>
                        <xsd:attribute name="name" type="xsd:string" use="required"/>
                        <xsd:attribute name="value" type="xsd:string" use="required"/>
                      </xsd:complexType>
                    </xsd:element>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="categories" minOccurs="0">  
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="category" maxOccurs="unbounded" minOccurs="0">  
                <xsd:complexType>
                  <xsd:attribute name="name" type="xsd:string" use="required"/>
                  <xsd:attribute name="eid" type="xsd:integer" use="required"/>
                  <xsd:attribute name="compid" type="xsd:integer" use="required"/>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="tags" type="xsd:string" minOccurs="0"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
<xsd:element name="filters" minOccurs="0">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="filter" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:string">
              <xsd:attribute name="name" type="xsd:string" use="required"/>
              <xsd:attribute name="operator" type="xsd:string" use="required"/>
              <xsd:attribute name="type" type="xsd:string" use="required"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:schema>

33.9.3.3 engineInstanceRequest Request XSD

The request XML for the `getEngineInstances(String userName, String engineInstanceRequest)` method is based on the `engineInstanceRequest request XSD`, shown in Example 33-28.

The `engineInstanceRequest request XSD` describes the set of rules that the request XMLs must follow to be valid. An example of a valid request XML for `getEngineInstances()` is:

```xml
<request>
  <enginetypeid>-1</enginetypeid>
  <locale>en_us</locale>
  <scope>LOCAL</scope> /*use GLOBAL for global scope*/
</request>
```

Example 33-28  engineInstanceRequest Request XSD

```xml
<?xml version="1.0" encoding="US-ASCII" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.example.com"
  targetNamespace="http://www.example.com"
  elementFormDefault="qualified">
  <xsd:element name="request">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="enginetypeid"/>
        <xsd:element name="locale" type="xsd:string"/>
        <xsd:element name="scope" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```
33.9.4 Understanding Web Service Response XSDs

For each of the web service calls, if the call is successful an XML string is returned. If the web service call results in an error, a service error XML string is returned. The client can then call the ServiceUtil() class to deserialize the XML string into ECSF runtime metadata objects or deserialize the service error XML string into an exception message.

When the query web service requests are successful, a response message is returned. The XML strings for successful web service calls are based on the XSDs for the following methods:

- getSavedSearch()
- getSavedSearches()
- saveSearch()
- deleteSearch()
- getSavedSearchDetails
- search()
- getEngineInstances()

If any of the query web service requests result in an exception, the exception is wrapped in a service error XML response message. The service error XML strings are based on the following XSD:

```xml
<?xml version="1.0" encoding="US-ASCII" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.example.com"
    targetNamespace="http://www.example.com"
    elementFormDefault="qualified">
    <xsd:element name="serviceError">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="code" type="xsd:string" minOccurs="0"/>
                <xsd:element name="message" type="xsd:string"/>
                <xsd:element name="stack" type="xsd:string"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

33.9.4.1 getSavedSearch()

The web service response message for the method String getSavedSearch(String userName, String savedSearchRequest) is based on the following XSD:

```xml
<?xml version="1.0" encoding="US-ASCII" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.example.com"
    targetNamespace="http://www.example.com"
    elementFormDefault="qualified">
    <xsd:element name="savedSearch">
        <xsd:complexType>
            <xsd:sequence>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```
33.9.4.2 getSavedSearches()

The web service response message for the method `String getSavedSearches(String userName, String savedSearchRequest)` is based on the following XSD:

```xml
<?xml version="1.0" encoding="US-ASCII" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns="http://www.example.com"
         targetNamespace="http://www.example.com"
         elementFormDefault="qualified">
    <xsd:include schemaLocation="saved-search.xsd"/>
    <xsd:element name="savedSearches">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="savedSearch" maxOccurs="unbounded" minOccurs="0"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

33.9.4.3 saveSearch()

The web service response message for the method `String saveSearch(String userName, String savedSearchRequest)` is the same as the response for `getSavedSearch()`. For information, see `getSavedSearch()`.

33.9.4.4 deleteSearch()

The web service response message for the method `String deleteSearch(String userName, String savedSearchRequest)` is the string `success`.

33.9.4.5 getSavedSearchDetails

The web service response message for the method `String getSavedSearchDetails(String userName, String savedSearchRequest)` is based on the following XSD:

```xml
<?xml version="1.0" encoding="US-ASCII" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns="http://www.example.com"
         targetNamespace="http://www.example.com"
         elementFormDefault="qualified">
    <xsd:include schemaLocation="saved-search.xsd"/>
    <xsd:element name="savedSearchDetails">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="savedSearch" maxOccurs="unbounded" minOccurs="0"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```
About Using the External ECSF Web Service for Integration

33.9.4.6 search()

The web service response message for the method `String search(String userName, String queryMetaDataRequest)` is based on the following XSD:

```xml
<?xml version="1.0" encoding="US-ASCII" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.example.com"
    targetNamespace="http://www.example.com"
    elementFormDefault="qualified">
    <xsd:include schemaLocation="query-metadata.xsd"/>
    <xsd:element name="searchResults">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="hitsMetaData" minOccurs="0">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="query" type="xsd:string"/>
                            <xsd:element name="page" type="xsd:integer"/>
                            <xsd:element name="pages" type="xsd:integer"/>
                            <xsd:element ref="queryMetaData"/>
                            <xsd:element name="categories" minOccurs="0">
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element name="category" maxOccurs="unbounded">
                                            <xsd:complexType>
                                                <xsd:sequence>
                                                    <xsd:element name="searchableObject" maxOccurs="unbounded">
                                                        <xsd:complexType>
                                                            <xsd:attribute name="name" type="xsd:string" use="required"/>
                                                            <xsd:attribute name="id" type="xsd:integer" use="required"/>
                                                            <xsd:attribute name="lastTimeCrawled" type="xsd:integer" use="optional"/>
                                                        </xsd:complexType>
                                                    </xsd:element>
                                                </xsd:sequence>
                                            </xsd:complexType>
                                        </xsd:element>
                                    </xsd:sequence>
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```
About Using the External ECSF Web Service for Integration
33.9.4.7 getEngineInstances()

The web service response message for the method String getString(String userName, String engineInstanceRequest) is based on the following XSD:

```xml
<?xml version="1.0" encoding="US-ASCII" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://www.example.com"
    targetNamespace="http://www.example.com"
    elementFormDefault="qualified">
    <xsd:element name="engineInstances">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="engineInstance" maxOccurs="unbounded" minOccurs="0">
                    <xsd:complexType>
                        <xsd:sequence>
                            <xsd:element name="category" maxOccurs="unbounded" minOccurs="0">
                                <xsd:complexType>
                                    <xsd:sequence>
                                        <xsd:element name="searchableObjects" minOccurs="0">
                                            <xsd:complexType>
                                                <xsd:sequence>
                                                    <xsd:element name="searchableObject" maxOccurs="unbounded" minOccurs="0">
                                                        <xsd:complexType>
                                                            <xsd:sequence>
                                                                <xsd:element name="fieldDefs" minOccurs="0">
                                                                    <xsd:complexType>
                                                                        <xsd:sequence>
                                                                            <xsd:element name="fieldDef" maxOccurs="unbounded" minOccurs="0">
                                                                                </xsd:element>
                                                                            </xsd:sequence>
                                                                        </xsd:complexType>
                                                                    </xsd:complexType>
                                                                </xsd:element>
                                                            </xsd:sequence>
                                                        </xsd:complexType>
                                                    </xsd:element>
                                                </xsd:sequence>
                                            </xsd:complexType>
                                        </xsd:element>
                                    </xsd:sequence>
                                </xsd:complexType>
                            </xsd:element>
                        </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```
<xsd:complexType>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="binding" type="xsd:string" use="required"/>
  <xsd:attribute name="dataType" type="xsd:string" use="required"/>
  <xsd:attribute name="isStored" type="xsd:boolean" use="required"/>
  <xsd:attribute name="isSecure" type="xsd:boolean" use="required"/>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
33.9.5 About Invoking the ECSF Web Service

Invoke the ECSF web service by creating a JAX-WS proxy using the Oracle JDeveloper Web Service Proxy wizard. Then, modify the client Java class named AppModuleSearchServiceSoapHttpPortClient to add calls to the ECSF web service.

Note:
It is assumed that the web service is deployed and running on a remote Oracle WebLogic Server instance.

33.9.5.1 Create a JAX-WS Web Service Proxy

Creating a JAX-WS web service proxy using the Oracle JDeveloper Create Web Service Proxy wizard generates all classes and Java files for the web service enabled methods under the package oracle.ecsf.service.query.common.

To create a JAX-WS web service proxy:

1. In the Application Navigator, right-click the project in which you want to create the web service proxy, and choose New.

2. In the New Gallery, expand Business Tier, select Web Services and then Web Service Proxy, and click OK.

3. On the Select Web Service Description page of the wizard, enter the URL for the WSDL that was generated when the application was deployed on the Oracle WebLogic Server, for example, http://myhost.us.example.com:7101/CrmSearchService/AppModuleSearchService?WSDL, then tab out of the field.

   If the Next button is not enable, click Why Not? to understand what problem JDeveloper encountered when trying to read the WSDL document. If necessary, fix the problem after verifying the URL and repeat this step.

   When the wizard displays an enabled Next button, then Oracle JDeveloper has recognized and validated the WSDL document.

4. For Specify Default Mapping Options, enter or choose a Java package name for the generated web service proxy class.

5. Click Next.

6. Continue through the pages of the wizard to specify details about the web service proxy. For more information about each page of the wizard, press F1 or click Help.

7. Click Finish.

8. Copy the following files from the system11.xxx/DefaultDomain/config/fmwconfig Oracle WebLogic Server directory to the JAX-WS Client application's META-INF directory.
   - jps-config.xml
   - cwallet.sso
About Modifying the AppModuleSearchServiceSoapHttpPortClient Class

After you create the JAX-WS proxy, modify the AppModuleSearchServiceSoapHttpPortClient class to include the calls to web service methods. Example 33-29 shows an example of a modified class.

The sample client class illustrates how each of the methods are called and how the XMLs are deserialized.

Example 33-29  Sample Client Class

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Properties;
import javax.xml.namespace.QName;
import javax.xml.ws.BindingProvider;
import oracle.ecsf.SearchException;
import oracle.ecsf.SearchHits;
import oracle.ecsf.client.SearchGroup;
import oracle.ecsf.client.dataobject.SavedSearch;
import oracle.ecsf.client.dataobject.SavedSearchDetails;
import oracle.ecsf.impl.QueryMetaDataImpl;
import oracle.ecsf.meta.MetaEngineInstance;
import oracle.ecsf.service.query.common.AppModuleSearchService;
import oracle.ecsf.service.query.common.AppModuleSearchService_Service;
import oracle.ecsf.service.query.ws.marshaller.ServiceUtil;
import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

public class AppModuleSearchServiceSoapHttpPortClient {
    private AppModuleSearchService_Service appModuleSearchServiceService;
    AppModuleSearchService appModuleSearchService;

    String wlsHost = null;
    String wlsPort = null;
    String wlsPassword = null;
    final Long engineInstanceId = 1L;

    public AppModuleSearchServiceSoapHttpPortClient() {
        initialize();
    }

    public void initialize() throws Exception {
```
if (System.getProperty("oracle.ecsf.debug.jdev") == null)
{
    System.setProperty("oracle.security.jps.config", findFile("META-INF/jps-config.xml"));
}
try
{
    // calling method initializeWLSparams to read parameter from file
   .ecsf.properties
    initializeWLSparams();
    URL aURL = new URL("http://" + wlsHost + ":" + wlsPort + "/CrmSearchService/AppModuleSearchService?WSDL");
    QName qName = new QName("/oracle/ecsf/service/query/common/", "AppModuleSearchService");

    AppModuleSearchServiceService = new AppModuleSearchService_Service(aURL, qName);
    AppModuleSearchService = AppModuleSearchServiceService.getAppModuleSearchServiceSoapHttpPort();

    ((BindingProvider)appModuleSearchService).getRequestContext().put(BindingProvider.USE_RNAME_PROPERTY, "weblogic");
    ((BindingProvider)appModuleSearchService).getRequestContext().put(BindingProvider.PASSWORD_PROPERTY, wlsPassword);
}
catch (Exception e)
{
    e.printStackTrace();
    System.out.println("Excpetion " + e);
}

/**
 * invoke the getEngineInstances() method exposed in WebService.
 */
public void testGetEngineInstance()
{
    ServiceUtil sUtil = new ServiceUtil();
    try
    {
        String query = "<request><enginetypeid>2</enginetypeid><locale>en-US</locale><scope>LOCAL</scope></request>";
        String response = appModuleSearchService.getEngineInstances("weblogic", query);
        Collection<MetaEngineInstance> instances = sUtil.toEngineInstances(response);
    }
    catch (Exception e)
    {
    }
}

/**
 * invoke the search() method exposed in WebService.
 */
public void testSearch()
{
    ServiceUtil sUtil = new ServiceUtil();
}
String request = "<request><queryMetaData><query><![CDATA[*]]></query><page>1</page><lang>en</lang><pageSize>10</pageSize><categories><category name="EmpSearch" eid="100010024205617" compid="100010026129681"></category></categories><queryMetaData><request>";
    try
    {
        String response = appModuleSearchService.search("weblogic", request);
        SearchHits hits = sUtil.toSearchHits(response);
    }
    catch (Exception e)
    {
    }
    
    /**
     * invoke the getSavedSearch() method exposed in WebService.
     */
    public void testGetSavedSearch()
    {
        ServiceUtil sUtil = new ServiceUtil();
        try
        {
            String request = "<request><name>TestServiceproxy</name><callerctx>GLOBAL</callerctx><locale>en-US</locale><scope/></scope></request>";
            String response = appModuleSearchService.getSavedSearch("weblogic", request);
            SavedSearch ss = sUtil.toSavedSearch(response);
        }
        catch (Exception e)
        {
        }
    }
    
    /**
     * invoke the getSavedSearches() method exposed in WebService.
     */
    public void testGetSavedSearches()
    {
        ServiceUtil sUtil = new ServiceUtil();
        try
        {
            String request = "<request><name></name><callerctx>LOCAL</callerctx><locale>en-US</locale><scope>LOCAL</scope></scope></request>";
            String response = appModuleSearchService.getSavedSearches("weblogic", request);
            ArrayList<SavedSearch> savedSearches = sUtil.toSavedSearches(response);
        }
        catch (Exception e)
        {
        }
    }
    
    /**
     * invoke the savedSearch() method exposed in WebService.
     */
    public void testSaveSearch()
    {
        ServiceUtil sUtil = new ServiceUtil();
        try
        {
            SavedSearch savedSearch = createSavedSearch("TestServiceproxy");
        }
String request = sUtil.toString(savedSearch, true);
appModuleSearchService.saveSearch("weblogic", request);
}
catch (Exception e)
{
}
}

/**
 * invoke the getSavedSearchDetails() method exposed in WebService.
 */
public void testGetSavedSearchDetails()
{
ServiceUtil sUtil = new ServiceUtil();
try
{
String ssRequest = "<request><name>TestServiceproxy</name><callerctx>GLOBAL</callerctx><locale>en-US</locale><scope></scope></request>";
String ssResponse = appModuleSearchService.getSavedSearch("weblogic", ssRequest);

String request = "<request>" + ssResponse + "<callerctx>null</callerctx><locale>en_us</locale><scope>LOCAL</scope></request>";
String response = appModuleSearchService.getSavedSearchDetails("weblogic", request);
SavedSearchDetails ssDetails = sUtil.toSavedSearchDetails(response);
}
catch (Exception e)
{
}
}

/**
 * invoke the deleteSearch() method exposed in WebService.
 */
public void testDeleteSavedSearch()
{
try
{
String ssRequest = "<request>QA<name>TestServiceproxy</name><callerctx>GLOBAL</callerctx><locale>en-US</locale><scope></scope></request>";
String ssResponse = appModuleSearchService.getSavedSearch("weblogic", ssRequest);

String request = "<request>" + ssResponse + "<callerctx>null</callerctx><locale>en_us</locale><scope>LOCAL</scope></request>";
String response = appModuleSearchService.deleteSearch("weblogic", request);
}
catch (Exception e)
{
}
}

private SavedSearch createSavedSearch(String sSearchName) throws SearchException
{
QueryMetaDataImpl queryMetaData = new QueryMetaDataImpl();
queryMetaData.setQueryString("%");
queryMetaData.setPageSize(10);
queryMetaData.setCurrentPage(1);
queryMetaData.setQuery("weblogic", request);
return queryMetaData;
}

Advanced Topics for ECSF 33-69
SearchGroup[] sgs = new SearchGroup[] { new SearchGroup("EmpSearch", "EmpSearch", engineInstanceId, null, null) }; queryMetaData.setSearchGroups(sgs);

SavedSearch savedSearch = new SavedSearch(SavedSearch.VAL_INVALID_ID, "weblogic", null, null, null, sSearchName, "search created for WS interop", null, queryMetaData.getQueryString(), false, "GLOBAL");

SavedSearchDetails ssd = new SavedSearchDetails(1L, "weblogic", null, null, null, SavedSearch.VAL_INVALID_ID, queryMetaData, null);
savedSearch.setSavedSearchDetails(ssd);

return savedSearch;
}

private void initializeWLSParams()
{
    //load wls host/port and ses wls host/port from ecsf.properties
    Properties prop = new Properties();
    FileInputStream fis;
    try
    {
        fis = new FileInputStream(System.getenv("T_WORK") + File.separator + "ecsf.properties");
        prop.load(fis);
        fis.close();

        wlsHost = prop.getProperty("WLS_HOST");
        wlsPort = prop.getProperty("WLS_PORT");
        wlsPassword = prop.getProperty("WLS_PWD");
    }
    catch (FileNotFoundException e)
    {
        System.out.println("File not found exception : " + e.getMessage());
    }
    catch (IOException e)
    {
        System.out.println("IO exception : " + e.getMessage());
    }
}

private String findFile(String findme)
{
    ClassLoader loader = Thread.currentThread().getContextClassLoader();
    URL u = loader.getResource(findme);
    Stringfullname = u.getFile();
    return fullname;
}

### 33.10 About Implementing Coherence Cache for Search

ECSF uses caches for metadata and configurations in support of performance and Coherence cache across servers in a cluster. When an object in the cache is updated, or when the cache is cleared on one server, the caches used by the same applications on all servers in the cluster must reflect these changes. This allows for the use of an administration management console to manage the state of the applications across servers in the cluster without having to restart them.
33.10.1 About Implement Cache Coherence for Search

Set the value in the `oracle.ecsf.cache.type` system property to indicate the type of caching that ECSF should use. The value can be either `local`, in which case ECSF uses a local cache, or `coherence`, in which case ECSF uses a distributed cache that is shared by multiple ECSF servers. Setting the value to `local` causes the `LocalCacheFactory` to be used to instantiate non-Coherence local caches. To enable Coherence caches, specify `coherence` as the value.

By default, the coherence caches are created with the `ecsf-coherence-cache-config.xml` file that is included in `ecsf.jar`. However, you can set the `oracle.ecsf.coherence.cache.config` system property to the path of an alternate coherence configuration file to use. The configuration file must have a mapping for cache names of the format `ecsf-_*`. In the default `ecsf-coherence-cache-config.xml` file, this name maps to a configuration of a Coherence Near Cache. For more information, see Developing Applications with Oracle Coherence. <<Link to COHDG.>>

33.11 About Localizing ECSF Artifacts

When developing applications for international users, it is necessary to customize the display of data by adding locale-specific components and translating text.

Locales identify a specific language and geographic region. You must customize how ECSF presents and formats messages (that is, the data displayed to users) based on their locale. Locale affects:

- Messages that you create during design time (for example, facet display names) and are displayed to the users who perform the queries.
- Messages you create during deployment (for example, searchable object names, search category names, and index schedule names displayed during crawl management).
- Crawlable content.

ECSF must determine the locale to display the messages in the appropriate language for the user.

33.11.1 How to Translate Strings in Groovy Expressions

ECSF provides a way for you to use translated strings to define the `Title`, `Body`, and `Keywords` properties of searchable objects.

The `ECSFFormat` class provides two functions that can be called in Groovy expressions:

- `format(String key, Object param1, Object param 2,...)` takes in a resource bundle key name, as well as up to ten parameters, to be added to the formatted string. This function allows you to use translatable strings in the Groovy expression for the `Title`, `Body`, and `Keywords` properties.
- `getLabel(String attributeName)` takes in an attribute name and gets the translated label for this attribute. This function provides a way for you to get the translated value of an attribute label.

ECSF also reserves a Groovy entity formatter that provides two functions:
formatter.format(String key, Object param1, Object param2,...) allows you to format a string based on a template.

formatter.getLabel(String attributeName) can be used to retrieve the UI label defined for an attribute.

To use translated strings in Groovy expressions, you must:

1. Associate a resource bundle to the view object.
2. Use the format() function in the Groovy expressions for the Title, Body, and Keywords properties.
3. Associate a translated label to the attribute.
4. Use the getLabel() function in the Groovy expressions for the Title, Body, and Keywords properties.

33.11.1.1 Associate Resource Bundles to View Objects

To define translatable strings for use as part of the crawlable search properties, you must associate a resource bundle to the view object and define the translatable strings on the resource bundle.

To add resource bundles, do the following:

1. In the overview editor, select the General navigation tab.
2. Click the Add icon in the Custom Properties table header and select Translatable Property to open the Select Text Resource dialog.
3. Select a resource bundle from the Resource Bundle dropdown menu.

   Note:
   If there is no resource bundle associated with the view object, then an external resource bundle is created in the application. Save the new resource bundle externally.

4. In the Display Value field, enter a string to associate with the key in the page’s resource bundle. For example, enter Identification Number: {0}. {1} {2} {3} {4} {5} {6}.

5. In the Key field, enter a string to uniquely identify a locale-specific object in the resource bundle. For example, enter EMPVIEW_BODY.

6. In the Description field, enter a description of any length for the key and value pair. For example, enter body value for crawling.

7. Click Save and Select.

   A new entry is added to the resource bundle, and the resource bundle is associated to the view object.

33.11.1.2 About Using the format() Function in Groovy Expressions

To use the translatable string in the Groovy expression for Title, Body, and Keywords, you can write a Groovy expression to call the formatter.format() function with
the key to the resource bundle entry as an input parameter. The other input parameters are standard Groovy input parameters.

For example, if the EMPVIEW_BODY translatable string is defined as:

EMPVIEW_BODY=Identification Number: {0}. {1} {2} {3} {4} {5} {6}

then you can use the following Groovy expression for the Body property:

formatter.format("EMPVIEW_BODY", Deptno, Empno ((DeptView.size() > 0) ? "Dept:" : ""), DeptView.Dname, ((StatesEAView.size() > 0) ? "State:" : ""), StatesEAView.Name, TestTransient)

The resource bundle key name, as well as up to ten parameters are added to the formatted string.

For date attributes, you can write a Groovy expression to call formatter.format(String attributeName), without a resource bundle key as an input parameter, to correctly format the date attribute value as part of the Groovy expression.

### 33.11.1.3 Associate Translated Labels to Attributes

In order for an attribute's label to be displayed in the correct locale, you must associate a resource bundle entry to the attribute.

To add labels to attributes, do the following:

1. In the overview editor, select the **Attributes** navigation tab.
2. Select the desired attribute in the Attributes table and click the **Edit selected attribute(s)** icon.
3. In the Edit Attribute dialog, select **Control Hints** to open the Control Hints page.
4. Click the icon next to the **Label Text** field to open and use the Select Text Resource dialog.
5. Select a resource bundle from the **Resource Bundle** dropdown menu.

**Note:**

If there is no resource bundle associated with the view object, then an external resource bundle is created in the application. Save the new resource bundle externally.

6. In the **Display Value** field, enter a string to associate with the key in the page's resource bundle.
7. In the **Key** field, enter a string to uniquely identify a locale-specific object in the resource bundle.
8. In the **Description** field, enter a description of any length for the key and value pair.
9. Click **Save and Select**.

A new entry is added to the resource bundle, and the resource bundle is associated to the view object attribute.
33.11.4 About Using the getLabel() function in Groovy Expressions

In order for the translated value of an attribute label to be crawled in the correct locale, the function `getLabel()` must be called as part of the Groovy expression for Title, Body, and Keywords, as shown in Figure 33-7.

**Figure 33-7  Example of Calling the getLabel() Function**

<table>
<thead>
<tr>
<th>Primary Table</th>
<th>Fusion.Emp.Emp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td><code>formatter.getLabel(&quot;Ename&quot;) + &quot; - &quot; + Ename + &quot; - &quot; + Sql</code></td>
</tr>
<tr>
<td>Body</td>
<td><code>formatter.format(&quot;EMPLOYEE_GBO&quot;, {Display, Empno ${(Depth*2)} &quot;* Depth:&quot; &quot;&quot;, DepthView Ename, (StakedEAlke...</code></td>
</tr>
<tr>
<td>Keywords</td>
<td><code>*Employee Department Job Salary Data &quot; + curdoc.getChldDoc(&quot;StatesEAView&quot;) + &quot; + &quot; + curdoc.getChldDoc(&quot;State...</code></td>
</tr>
</tbody>
</table>

In the example, `formatter.getLabel("Ename")` retrieves the translated value of the attribute's label from the view object definition.

Calling the `getLabel()` function retrieves the translated value of the attribute's label from the view object definition for crawling.

33.11.2 About Localizing Facet Display Names

If not previously loaded for the current user's locale, facet display values are loaded by querying the list of values (LOV) associated with the facet in the context of the current user at query time. Whatever values or display values returned by querying the LOV in the context of the current user are loaded and returned as part of the facet in the query result for display in the user interface.

There are many ways to configure LOVs for localization. Two of them are described here: using the VL table and using resource bundles.

33.11.2.1 About Configuring LOVs for Localization Using the VL Table

Facets are displayed in a tree.

*Facet Display Name
  *Facet Entry Display Name
  *Facet Entry Display Name
  ...

The facet display name is translated per the resource bundle supported by Oracle Application Development Framework (Oracle ADF). The facet entry display names are translated per the LOV definition with context locale binding.

For example:

*Organization
  *Vision USA
  *Vision Canada

Organization is translated via the Oracle ADF resource bundle with key ORGANIZATION_ID.
To translate Vision USA and Vision Canada, you need to create a view object based on the VL table that supports localization, for example, HR_ALL_ORGANIZATION_UNITS_VL.
After this view object is created and used as the LOV for the BUSINESS_UNIT_ID of the base searchable object, the value for the field name is pulled from the VL table with the correct locale of the current user. For information on how to create facets, see How to Implement Faceted Navigation.

When your facet is localized, Oracle Fusion Applications Search UI requests facet definitions from ECSF when users performs a search. ECSF will load the facet definition, including facet names from the resource bundle based on user's locale. When the user clicks the facet, ECSF will execute the LOV for the facet and retrieve the data from the database based on the current user's language setting. In the example, the organization name will be displayed to the user in his language. When the user clicks on a particular node, such as Vision USA, ECSF will perform a query against the search engine with a filter on Organization ID = 204 where 204 is the organization ID for Vision USA.

33.11.2.2 About Configuring LOVs for Localization Using the Resource Bundles

Facet display names can be localized with Oracle ADF resource bundles. When you create a facet, specify the name of a facet by referring to a resource key. At runtime, when the resource bundle is translated, the correct display name will be retrieved based on the user's locale. Localization of facet entries are supported by view object based LOVs. The view object must be designed to support localization.

**Note:**

Since facets are cached for performance, internationalized messages must be cached with the locale.

If the display name is not available for the current locale, the facet name is used.
ECSF uses LOVs to support facets, and the LOV display values must be translatable. The data sources of the LOVs used for facets can be view objects with static data or view objects with dynamic data (pulled from database through SQL). Display values of LOVs whose data sources are view objects with static data are displayed using the resource file corresponding to the application’s current locale. Display values of LOVs populated with dynamic content can be translated based on the locale in the user’s context.

33.11.2.2.1 How to Translate LOV Display Values

You can translate the LOV display values based on the locale in the user’s context by adding a language code column to the view object. The following instructions are based on the example provided in How to Implement Faceted Navigation. It assumes that you have an EmpView base view object with a working LOV on the StatesView view object already defined.

To translate the LOV display values, do the following

1. Add a **LANG** column (VARCHAR2) to the **STATES** table.

2. Populate the **LANG** column for all rows with a 2-character ISO639 code (for example, **en** or **fr**).

3. Add the **LANG** column to the States entity object and StatesView view object. Alternatively, you can re-create it from scratch.

4. Create a view criteria on StatesView (on the view object, navigate to Query > View Criteria > + and click the Add Criteria button) with the following values:
   - **Attribute**: Lang
   - **Operator**: equal to
   - **Operand**: Bind Variable

5. For Parameter, click **New** to create a bind variable with the following values:
   - **Name**: UserLang
   - **ValueType**: Expression

6. Click OK, and click OK again.

7. Navigate to EmpView > View Accessors, select StatesView1, and click **Edit**.

8. In the Edit View Accessor dialog, select the view accessor (StatesViewCriteria) in the Available list and click the Move icon to add it to the Selected list.

9. Edit the **UserLang** bind variable from StatesView by entering the following Groovy expression that evaluates to the user’s current language:

   ```groovy
   adf.context.locale.getLanguage()
   ```

The view criteria (similar to a defined WHERE clause) on StatesView and a bind variable named **UserLang** are created. The EmpView base view object is configured to use the view criteria when querying StatesView and the value of the bind variable is set to the user’s current language. During runtime, only the
records where LANG matches the locale.getLanguage() of the locale is returned.

### 33.11.3 About Localizing Crawl Management Display Names

Crawl management display names—that is, the searchable object names, search category names, and index schedule names displayed during crawl management—must be translated to the language of the administration tool user. Enabling translation for these display names is handled in the database through the translation table so that it can be shared across runtime, the ECSF Command Line Administration Utility, and Fusion Applications Control.

**Note:**

For Fusion Applications Control, the display name for searchable objects, search categories, and index schedules is in the language of the object created by the administration users.

Deployment to Oracle SES is not based on translated fields.

Localize the crawl management display names by translating the seed data that you create in English. For information see [Initializing Oracle Fusion Application Data Using the Seed Data Loader](#).

### 33.11.4 About Localizing Crawlable Dynamic Content

Content crawled into Oracle SES can support different languages. For searchable objects with dynamic content (content pulled from a database through view objects), ECSF supports one language per business object instance. An example is a sales order created by a customer with a particular language. In other words, ECSF only supports language on a per record or row basis.

While ECSF supports only one language per object instance, each instance can be a different language. ECSF detects the language for a given document. For information, see [About Determining Locale](#).

### 33.11.5 About Localizing Crawlable Template Content

In addition to dynamic content, searchable objects can contain template content. Templates are text formats you create to form the body, title, keywords, action titles, parameters, and so on. In ECSF, you use Groovy scripting language to create expressions for some fields. The language must be identified to translate Groovy-enabled fields into the appropriate language before they are sent to the search engine. Localization for Groovy-enabled fields is supported by formatting functions.

ECSF provides a set of formatting functions based on Java Text Format Function. These formatting functions allow you more flexibility to build your desired formats. [Example 33-30](#) illustrates the use of a Groovy function with Java Text Formatting to form the body for localization.

EMPLOYEE_TEMPLATE and REPORTS_TEMPLATE are keys to two translatable templates that are stored in a resource bundle. The template must be a legal format string for java.text.MessageFormat. (FirstName, LastName, EmailAddress, and ZipCode refer to view object attributes.)
To illustrate how the expression is resolved during runtime, see Table 33-2 for a sample view object, Table 33-3 for a sample child view object, and Example 33-31 for a sample resource bundle.

**Table 33-2 Sample Parent View Object (Employee)**

<table>
<thead>
<tr>
<th>ID</th>
<th>FirstName</th>
<th>LastName</th>
<th>EmailAddress</th>
<th>ZipCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>Wayne</td>
<td><a href="mailto:john.wayne@example.com">john.wayne@example.com</a></td>
<td>94065</td>
</tr>
<tr>
<td>2</td>
<td>Chris</td>
<td>Smith</td>
<td><a href="mailto:chris.smith@example.com">chris.smith@example.com</a></td>
<td>94066</td>
</tr>
</tbody>
</table>

**Table 33-3 Sample Child View Object (Reports)**

<table>
<thead>
<tr>
<th>ParentID</th>
<th>FIRST_NAME</th>
<th>LAST_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>John</td>
<td>Doe</td>
</tr>
<tr>
<td>1</td>
<td>Jake</td>
<td>Ray</td>
</tr>
<tr>
<td>2</td>
<td>Bob</td>
<td>Barley</td>
</tr>
<tr>
<td>2</td>
<td>Chris</td>
<td>Carpenter</td>
</tr>
</tbody>
</table>

**Note:**

This example is strictly used to illustrate functionality. Concatenation of words using two resource messages is not translation friendly.

The following shows the resulting data sent to Oracle SES during runtime:

Employee John Wayne with email address john.wayne@example.com works in zipcode 94065
direct report named John Doe has a direct report named Jake Ray

Employee Chris Smith with email address chris.smith@example.com works in zipcode 94066
direct report named Bob Barley has a direct report named Chris Carpenter

Resource bundles are used to store language variations, while translation is the responsibility of Oracle Fusion Applications i18n support infrastructure.

**Example 33-30 Using Groovy Function to Translate the Body**

```groovy
formatter.format("EMPLOYEE_TEMPLATE", FirstName, LastName, EmailAddress, ZipCode)
for (item in doc.getChildDocs("Reports"))
{
    formatter.format("REPORTS_TEMPLATE", item.fields.FIRST_NAME, item.fields.LAST_NAME);
}
```

**Example 33-31 Sample Resource Bundle**

```java
id=EMPLOYEE_MESSAGE,message="Employee {0} {1} with email address {2} works in zipcode {4}"
id=REPORTS_TEMPLATE,message=" direct reports {0} {1}"
```
33.11.6 About Determining Locale

To display messages in the appropriate language for the user, ECSF must determine the locale. How ECSF determines locale varies by module.

33.11.6.1 About the Search Page

The Search page, an extension of Oracle JDeveloper, uses the locale of the Java Virtual Machine (JVM). This locale is set at either the machine level or at JDeveloper level. For information about setting the JVM locale for JDeveloper, see the JDeveloper online help.

33.11.6.2 About the ECSF Command Line Administration Utility

The ECSF Command Line Administration Utility, a standalone and single user application, uses the default locale returned by `Locale.getDefault()`.

33.11.6.3 About Crawl

You can specify the locale of the data to be indexed by using any of the following methods:

- Language field
- Crawler’s language preference
- Java Virtual Machine (JVM) default locale

You can specify a view object attribute as a language field for a searchable object by using the Search page. For information, see Setting Search Property Values for View Objects.

The value of this field for each instance is the language for the instance.

If you do not specify a language field for a given searchable object, ECSF uses the language preference of the crawler user. This implementation is application specific. ECSF obtains a session user through identity manager, and obtains the locale for the user while crawling data. There must be a crawler user with a proper locale profile, including language preference, created for Oracle Fusion Applications. This user’s language preference is used as the default language if a language field does not exist.

If no crawler language is available, JVM's default locale (`Locale.getDefault()`) is used to identify the language, and the JVM default language is used.

33.11.6.4 About Query

The user who calls the ECSF query time API must bind the locale to `SearchContext`. If the locale is not set in the search context, ECSF attempts to get from `ADFContext`. If that fails, the default locale, shown in Example 33-32, is used.

Example 33-32  Default Locale for Searcher

```java
ADFContext adfContext = ADFContext.getCurrent();
Locale mLocale = (adfContext != null && adfContext.getLocale() != null) ? adfContext.getLocale() : Locale.getDefault();
```

33.12 About Using ECSF Diagnostics

This section outlines the use of the diagnostic tests available in the ECSF application. The diagnostic service can be reached in a browser by appending Diagnostics to the
ECSF data service URL. The data service URL will be shown on the Search Engine Instance Parameters tab in the Oracle Enterprise Manager Fusion Applications Control.

All test results will include the number of failures and errors, and the time taken in milliseconds to execute the test.

Tests that are marked as protected require the ECSF_All_Services permission to access.

### 33.12.1 Query Tests

These tests are used to diagnose issues in query functionality.

#### 33.12.1.1 About the Simple Query Test

This test can be used as a sanity check to ensure that queries are returning data. The test queries for all records from the selected category. Table 33-4 lists the parameters.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Searchable Group</td>
<td>Drop-down</td>
<td>The search category against which to invoke the category. The format of the parameter is &lt;category&gt;@&lt;engine instance&gt;. The selection [All Groups] can be made to search against all of the displayed groups.</td>
</tr>
</tbody>
</table>

Note: The engine instance SES proxy user is used for the search and the search keyword is "*".

**Result**

Indicates if any search hits and facets are returned. The actual data are not displayed due to security reasons.

#### 33.12.1.2 About the Searchable Object Metadata Test

This test provides information about the selected searchable object and validates groovy expressions for the underlying view object. Table 33-5 lists the parameter of this test.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Searchable Object</td>
<td>Drop-down</td>
<td>The search object to inspect. The format of the parameter is &lt;object&gt;@&lt;engine instance&gt;.</td>
</tr>
</tbody>
</table>

**Results**

- Searchable object configuration, including its display name, view object name, ID, schema name, registered engine instance, associated schedule, and associated categories.

- Searchable object status including if it is deployed, if it is active, and if the latest engine instance parameters have been applied to it. The deployment and crawl statuses of the associated schedule are also displayed, along with the deployment statuses of associated categories.
• View object information, including the driving table, plugin class, if the plugin is ACL enabled, view object SQL select statement, view object fields, and child references.

• Results of the body, title, keywords, and default action title validations.

• Display of top-level facets. Actual data are obfuscated due to security reasons.

• Searchable object actions.

33.12.1.3 About the Searchable Groups Test

This test displays the searchable groups for an engine instance. Table 33-6 lists the parameters of this test.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine Instance Name</td>
<td>Drop-down</td>
<td>Name of the engine instance.</td>
</tr>
</tbody>
</table>

Note: All searchable groups in the search application are displayed.

**Result**

Searchable groups for this engine instance, along with their associated searchable objects.

33.12.1.4 About the Advanced Query (Protected) Test

This test can be used as a more granular check to ensure that queries are returning data. Table 33-7 lists the parameters of this test.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Searchable Group</td>
<td>Drop-down</td>
<td>The search category to invoke the category against. The format of the parameter is &lt;category&gt;@&lt;engine instance&gt;. The selection [All Groups] can be made to search against all of the displayed groups.</td>
</tr>
<tr>
<td>Query Language (optional)</td>
<td>String</td>
<td>Specify a language for the query (language short code, such as &quot;en&quot;). If no language is specified then the currently configured language will be used.</td>
</tr>
<tr>
<td>User Name</td>
<td>String</td>
<td>The user name to search with. Only data visible to this user will be shown.</td>
</tr>
<tr>
<td>Query String</td>
<td>String</td>
<td>String to search for.</td>
</tr>
</tbody>
</table>

**Results**

• Result hit count: The number of hits for the first page of results (up to 10).

• Estimate hit count: The estimated total hit count will for this query.

• Alternate keywords if any.

• Top-level facets including counts.
• Search results: The title, body, and attribute values for the first page of results (up to 10 records) will be shown.

### 33.12.2 About Crawl Tests

These tests are used to diagnose issues in crawler functionality, including ECSF feed generation.

#### 33.12.2.1 About the Crawl Searchable Object Test

This tests crawling of the object, splitting of its table, and processing of data feeds. Table 33-8 lists the parameters of this test.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Searchable Object</td>
<td>Drop-down</td>
<td>The search object to crawl. The format of the parameter is <code>&lt;object&gt;@&lt;engine instance&gt;</code>.</td>
</tr>
<tr>
<td>Incremental Crawl</td>
<td>Checkbox</td>
<td>Tests incremental crawl on the Searchable Object, otherwise a full initial crawl will be simulated.</td>
</tr>
<tr>
<td>Number of Batches to test</td>
<td>Input (integer)</td>
<td>Number of data feeds to execute (such as batches). -1 can be specified to process all data feeds/batches (not recommended for large data sets).</td>
</tr>
</tbody>
</table>

Note: The engine instance crawl user is used for this test.

**Results**

• SQL statement for the searchable object.

• Statistics on each work unit (batch) including the number of documents crawled, the time taken, and the rate of documents processed by ECSF.

• Total statistics for all work units processed.

#### 33.12.2.2 About the SES Instance Test

This test displays information about the engine instance, tests proxy login into SES, tests ECSF security service authentication, and displays ECSF metadata as well as SES metadata. Table 33-9 lists the parameter of this test.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine Instance Name</td>
<td>Drop-down</td>
<td>Select the ECSF engine instance to test.</td>
</tr>
</tbody>
</table>

**Results**

• SES admin endpoint.

• ECSF service endpoint.

• Result of proxy login into SES with stored credentials.

• ECSF security service authentication status.
• Search groups in ECSF.
• Source groups in SES.
• Index schedules in ECSF including the statuses in ECSF.
• Index schedules in SES including the statuses in SES.
• Searchable objects in ECSF.
• Data sources in SES.
• Number of configured proxy logins (federated trusted entities) in SES.
• Configured identity plugin URL in SES.

33.12.2.3 About the Control Feed Test
This tests splitting of the table and generates the complete control feed for a searchable object. Table 33-10 lists the parameters of this test.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Searchable Object</td>
<td>Drop-down</td>
<td>Select the searchable object.</td>
</tr>
<tr>
<td>Incremental Crawl</td>
<td>Checkbox</td>
<td>If checked, it tests incremental crawl on the searchable object; otherwise a full initial crawl will be simulated.</td>
</tr>
</tbody>
</table>

Note: The engine instance crawl user is used for this test.

Result
Complete control feed (including list of data feeds).

33.12.2.4 About the Data Feed (Protected) Test
Tests a data feed for a searchable object. This tests crawling of the object, splitting of its table, and processing of data feeds. Table 33-11 lists the parameters of this test.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Feed URL</td>
<td>Input (String)</td>
<td>Enter a full data feed URL for a searchable object.</td>
</tr>
</tbody>
</table>

Note: The part of the URL that points to the ECSF data service will be ignored. Instead the data feed will be invoked against the data service stored in the database for the object’s associated engine instance.

Result
Complete XML data feed.

33.12.3 About Environment and Configuration Information Tests
These tests provide information about the configuration of the ECSF environment.
33.12.3.1 About the Configuration Parameters Test

This test lists the ECSF search engine instance parameters. Table 33-12 lists the parameter of this test.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine Instance</td>
<td>Drop-down</td>
<td>Specify the engine instance to inspect.</td>
</tr>
</tbody>
</table>

**Result**

Values of the search engine instance parameters. Usernames will be obfuscated for security reasons.

33.12.3.2 About the Environment Information Test

This test retrieves environment information such as ECSF properties and ECSF jar extraction path on the server.

**Results**

- System properties that begin with "oracle.ecsf".
- Location of the Java class for this test on the server.

33.12.3.3 About the Data Source Test

This tests the connection to the SearchDB data source.

**Results**

- Displays if the application was able to obtain the connection successfully.
- If connection is successful displays the connection URL.

33.12.3.4 About the Application Extension/ApplCore Session Locale (Protected) Test

This tests the application extension session for the given user. Table 33-13 lists the parameter of this test.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Name</td>
<td>Input (String)</td>
<td>Specify the user for the test.</td>
</tr>
</tbody>
</table>

**Results**

- Initializes the application extension session for the user.
- Locale for the ApplCore session.

33.12.4 About Security Tests

These tests are used to diagnose issues in the security layer of the ECSF application.
33.12.4.1 About the Security (Protected) Test

This tests various security functionalities for a user. Table 33-14 lists the parameters of this test.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Name</td>
<td>Input (String)</td>
<td>Specify the user to test.</td>
</tr>
<tr>
<td>Permission (optional)</td>
<td>Input (String)</td>
<td>Specify a permission to check for the user. The format of the permission should be TYPE:NAME:ACTIONS.</td>
</tr>
<tr>
<td>Engine Instance Name</td>
<td>Drop-down</td>
<td>The engine instance is used when retrieving credentials for non APPID users.</td>
</tr>
</tbody>
</table>

**Results**

- Checks that the user exists in the system.
- Checks that the user has the given permission.
- Checks that the user has access to the oracle.ecsf credential map.
- Displays if the user has access to all ECSF services.
- Displays if the user has access to the Diagnostics service if it does not have access to all services.
- Checks that the credentials for the user can be obtained from the credential store.

33.12.4.2 About the Credential Store Test

This tests system access to the credential store. Table 33-15 lists the parameters of this test.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Engine Instance Name</td>
<td>Drop-down</td>
<td>The engine instance determines which map to access in the credential store.</td>
</tr>
</tbody>
</table>

**Result**

Checks that the system can write to the credential store by creating and removing credentials for a dummy user.

33.12.4.3 About the Security Plugin (Protected) Test

This tests the security plugin of a searchable object for a user. Table 33-16 lists the parameters of this test.
Table 33-16  Parameters for Security Plugin (Protected) Test

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Searchable Object</td>
<td>Drop-down</td>
<td>The search object to test against. The format of the parameter is &lt;object&gt;@&lt;engine instance&gt;.</td>
</tr>
<tr>
<td>User Name</td>
<td>Input (String)</td>
<td>Specify the user to test.</td>
</tr>
<tr>
<td>Attribute Name (optional)</td>
<td>Input (String)</td>
<td>The security attribute to use for the test. If no attribute is given then the default security attribute will be used.</td>
</tr>
</tbody>
</table>

Results

- Security attribute values for the user from the plugin.
- Security principals for the user.

33.13 About Troubleshooting ECSF

This section describes common problems that you might encounter when using ECSF and explains how to solve them.

33.13.1 Understand Common Problems and Solutions

The following are common problems you may encounter and solutions that solve them:

- Cannot See Data in Data Feeds
- Configuration or Data Feed Execution Thread Is Busy for Longer than the Configured Warning Timeout
- Class Not Found Errors When Running the ECSF Servlet
- Out of Memory Error when Deploying the ECSF Application to or Running the Application
- Blank /UI Shell Pages
- Memory Leak on ThreadLocal Variable (SearchContext)

33.13.1.1 Cannot Remove the ECSF Runtime Server Library

You remove the ECSF Runtime Server library from the project and save, but the library reappears in your project.

Problem:

The Common-Model.jar file that you added to your project contains a dependency on ECSF Runtime Server.

Solution:

Remove the Common-Model.jar file.

33.13.1.2 Cannot See Data in Data Feeds

Your feed request made through either Oracle SES or the browser returns no data.
Problem:
After you initially crawl the data source, subsequent feed requests result in incremental feeds (that is, feeds that contain only changes in data).

Solution:
Override the incremental feed by performing one of the following tasks:

- Through Oracle SES, start full indexing from the Fusion Applications Control. For information, see the "Starting Full Indexing" section in *Administrator’s Guide*.
- Through the browser, append `?forceInitialCrawl=true` to the config feed.

### 33.13.1.3 Configuration or Data Feed Execution Thread Is Busy for Longer than the Configured Warning Timeout

You receive an error indicating that the execution time of the configuration or data feed execution thread is exceeding the timeout settings, for example:

Error: name has been busy for "elapsedTime" seconds working on the request "curReq", which is more than the configured time (StuckThreadMaxTime) of "maxTime" seconds.

Following are two possible causes and solutions for this issue:

Problem:
A SQL script being executed by ECSF for the configuration feed or data feed is taking a long time to execute.

Solution:
Enable SQL tracing to find long-running SQL processes, and tune the SQL script to reduce the execution time.

Problem:
ECSF is unexpectedly running longer than the configured time.

Solution:
Increase the Oracle WebLogic Server warning timeout setting.

### 33.13.1.4 Class Not Found Errors When Running the ECSF Servlet

You receive Class Not Found errors when you run the ECSF servlet.

Problem:
When you created the ECSF application, you did not select the Fusion Web Application (ADF) template.

Solution:
Add the Fusion Web Application (ADF) template through Application Properties.

### 33.13.1.5 Out of Memory Error when Deploying the ECSF Application to Oracle WebLogic Server or Running the Application

You receive a java.lang.OutOfMemoryError: PermGen space exception when you deploy the ECSF application to Oracle WebLogic Server instance or when you run the application.

Problem:
MaxPermSize is set too low.
Solution:
Increase MaxPermSize by starting the WebLogic JVM using the –XX:MaxPermSize=256m parameter.

If you start the ECSF servlet from within JDeveloper using the Integrated WebLogic Server:

1. Go to the jdev cache directory /.jdeveloper/DefaultDomain/bin.
2. Open setDomainEnv.sh.
3. Locate the line that contains –XX:MaxPermSize=128m, and change it to –XX:MaxPermSize=256m.

33.13.1.6 Blank Oracle ADF/UI Shell Pages
Oracle ADF/UI Shell pages fail to load or a blank page appears. You receive the following exception in the JDeveloper Log window:

java.lang.NoClassDefFoundError: oracle/ecsf/client/SearchCtrl at oracle.apps.fnd.appcore.globalSearch.ui.ecsf.ECSFSearchUtils.getSearchControl(ECSFSearchUtils.java:67)

Problem:
The ECSF Client library is missing from the project class path.

Solution:
Update the project class path with the ECSF Client library.

33.13.1.7 Memory Leak on ThreadLocal Variable (SearchContext)
You encounter memory leaks while using the ECSF query API to define the Oracle Fusion Applications Search user interface.

Problem:
SearchContext is implemented as a ThreadLocal variable, which is created when ContextFactory.getSearchContext() is first called in the current Thread. All subsequent calls to getSearchContext() within the same Thread results in that instance being returned. Since the ThreadLocal variable remains associated with the Thread, memory leaks occur.

Solution:
After the completion of your Oracle Fusion Applications Search UI logic and before results are returned to the UI for rendering, you must call the SearchContext.release method.

When the SearchContext.release method is called on the SearchContext, the instance of the ThreadLocal is removed from the current Thread. A subsequent call to getSearchContext then results in the creation of a new instance of the SearchContext ThreadLocal variable. The SearchContext.release method also implicitly call releaseConnection().
Note:

If you call `SearchContext.release` and then need to call `getSearchContext()` again within the same Thread, then you must set any parameters you need on the `SearchContext` again as it is a brand new instance.

### 33.13.1.8 About Checking the Space Availability for SES Crawls in the Database

The table spaces required are:

- **SEARCH_DATA**
- **SEARCH_INDEX**
- **SEARCH_TEMP**

Verify the details for **SEARCH_DATA** using this SQL statement:

```sql
select FILE_NAME, TABLESPACE_NAME, BYTES, MAXBYTES, BLOCKS, MAXBLOCKS from sys.dba_data_files where TABLESPACE_NAME = 'SEARCH_DATA';
```

Use this command to add additional table space for **SEARCH_DATA**:

```sql
ALTER TABLESPACE SEARCH_DATA ADD DATAFILE '/slot/ems7248/oracle/db/apps_st/data/SEARCH_DATA_3.dbf' SIZE 1052M AUTOEXTEND ON;
```

The DBF filepath will depend on the environment being used. To learn the exact DBF file to be used, use the `select` command shown above. For example: `/slot/ems7248/oracle/db/apps_st/data/SEARCH_DATA_3.dbf`

To delete an additional datafile:

```sql
ALTER TABLESPACE SEARCH_DATA DELETE DATAFILE '/slot/ems7248/oracle/db/apps_st/data/SEARCH_DATA_9.dbf';
```

The same steps can be used to add additional data files for other table spaces.

### 33.13.1.9 About Crawling with a Different User

On a provisioned environment, the crawls will not work for all users. The users will need to be modified.

**Pre-requisite:** Find the application role for which the user must be added.

As an example, take the Application role `ZCA_CRM_FUSION_SEARCH_CRAWL_DUTY` and assign `sales_admin` to this role. `ZCA_CRM_FUSION_SEARCH_CRAWL_DUTY` is the application role that is necessary for a user to access the feeds.

- Login to CRM EM with the required credentials.
- Right-click CRMDomain and select **Security > Application Roles**.
- Select the Stripe as `crm` and in Roles enter `ZCA_CRM_FUSION_SEARCH_CRAWL_DUTY` and search.
- Click the role `ZCA_CRM_FUSION_SEARCH_CRAWL_DUTY` and click Add to add the `sales_admin` user.
- Click **OK**.
The user sales_admin now can be used for crawling.

### 33.13.1.10 "FND-6601 Search categories are not available"

**Problem:**
The endpoints for the search applications are not up and running.

**Solution:**
Set the correct search application endpoints in the `connections.xml` file of the client application. Make sure the `IS_ACTIVE` parameter corresponding to this search engine instance is set to "Y". Make sure the `IS_ACTIVE` parameter corresponding to this search engine instance is set to "Y" in the ECSF_PARAMETER table.

**Problem:**
If your search application only contains one category and the category is secured, the Global Search UI will display this error message if this category does not pass the permission check. In this case, no categories are available for search.

**Solution:**
Usually this is because the permission set on the searchable object is not accessible to the logged-in user, causing the secured category to not be returned by the search application.

**Problem:**
Search_Server1 is not up and running.

**Solution:**
Bounce if required.

**Problem:**
wsm-pm is not up and running. For example, wsm-pm is hosted in the SCM_CommonServer for FSCM.

**Solution:**
Bounce if required.

### 33.13.1.11 "FND-6603 Search is not currently available"

**Problem:**
The identity management setup in SES administration is pointing to an invalid URL combination and the proxy user cannot log into ECSF security service.

**Solution:**
Change the identity management setup to point to the correct url using the correct username and password.

**Problem:**
The response from the Search servers may be very slow and the Search UI is unable to fetch all the results.

**Solution:**
Increase the WS timeout using this parameter in the startup section on the WLS console for the Search Server

```
-Doracle.ecsf.service.ws.timeout=500000
```
33.13.1.12 "FND-6606 An application error occurred with this search"

Problem:
In B16, the category drop-down is not displayed in the main UI page. The Global Search UI does not load categories until an initial search is performed by clicking the Search button. The Global Search UI will show this error if getting categories failed against all the search applications. The endpoints for the search applications are not up and running or the search applications are inactive in the database.

Solution:
Set the correct search application endpoints in the connections.xml file of the client application. Make sure the IS_ACTIVE parameter corresponding to this search engine instance is set to "Y". Make sure the IS_ACTIVE parameter corresponding to this search engine instance is set to "Y" in the ECSF_PARAMETER table.

Cause:
If the search applications are up and running but this error is displayed, all the search applications probably are returning errors.

Look for the following line in the messages logged in the client server log, such as CustomerServer-diagnostics.log:

```java
at oracle.ecsf.service.query.ws.marshaller.ServiceUtil.toError(ServiceUtil.java:1499)
```

33.13.1.13 Query Does Not Return Search Results but No Errors Are Displayed on the UI

Cause:
The crawl for the data sources under this category was not successful.

Solution:
Log into SES administration and check that the schedule for the data source indexed successfully.

33.13.1.14 FUSION_RUNTIME.FND_TABLE_OF_VARCHAR2_4000 Exception on Schedules

Cause:
This exception may occur on the SES when a user tries to run a schedule. This is caused if the administration server and the manager servers are not started from the correct locations. (Console/node manager)

Solution:
The adminserver must be started using nodemanager and all other managed servers must be started by using the Administration console.

33.13.1.15 Where Can I Find the SES-ESS Crawler Logs?

On most of the provisioned environments, the ESS logs will be found in <APPLTOP>/instance/ess/rd. This will contain the crawl records for all the crawls run on the SES installed in that environment. If the location of these logs is changed, log in to the Enterprise Manager farm and update the RequestFileDirectory attribute.
33.13.1.16 My Crawls Are Failing
The SES crawls may fail due to various reasons. Some of the most common issues are listed here.

- Check if the ESS and search_server1 managed servers are running on the Common domain. If these are not running, the crawls will fail.
- Check if the corresponding Search server is in the "RUNNING" state.
- Check if the END points mentioned in the SES Source page are accessible.
- Check if the database has sufficient space to accommodate the crawled data. Check first query to increase the table space.
- Verify the security attributes for the Searchable View Object (SVO). Also verify if the passwords for CRAWL_APPIDs are correct.
- Check that these users are not locked:
  FUSION_APPS_FSCM_SES_CRAWL_APPID,
  FUSION_APPS_CRM_SES_CRAWL_APPID, and
  FUSION_APPS_HCM_SES_CRAWL_APPID.

33.13.1.17 How to Get the Password for the SES Administration Page
The SES Administration page uses "searchsys" as the default user for logging in. The password for SES can be discovered using WebLogic Server Scripting Tools (WLST) commands.

1. Connect to WLST.
2. Run the command:
   ```java
   listCred(map="oracle.apps.security",key="FUSION_APPS_ECSF_SES_ADMIN-KEY")
   ```
   Sample output:
   ```
   Already in Domain Runtime Tree
   [Name : searchsys, Description : null, expiry Date : null]
   PASSWORD:welcome1
   ```

33.13.2 About Diagnosing ECSF Problems
To diagnose ECSF problems in the development environment, you can view the log messages in the Oracle WebLogic Server at `DOMAIN_HOME/servers/SERVER_NAME/logs/SERVER_NAME-diagnostic.log`.

You can configure the log level for ECSF by using Fusion Applications Control or Oracle WebLogic Scripting Tool.

The log level for ECSF can be set to the following values:

- **TRACE** for FINE
- **NOTIFICATION** for INFO (default level)
- **WARNING** for WARNING
- **ERROR** for SEVERE

The following ECSF loggers can be used for logger name:
• oracle.ecsf.AdminLogger
• oracle.ecsf.ClientLogger
• oracle.ecsf.RuntimeLogger

Related Links
The following document provides more information about the topic discussed in this section:

• For more information, see the "Configuring Settings for Log Files" chapter in the
  Administering Oracle Fusion Middleware.
Initiate a SOA Composite from an Oracle ADF Web Application

This chapter describes what a user action or other activity in an Oracle ADF web application needs to do to invoke a SOA composite. The invocation is asynchronous and does not require a response. Inside the SOA composite, an Oracle Mediator component can provide routing and transformation, a BPEL component can provide business process orchestration, a human task service can provide workflows, and a decision service can provide complex business rules based decision making.

**When to implement:** A user action or other activity in an Oracle ADF web application needs to invoke a SOA composite. The invocation is asynchronous and does not require a response. Inside the SOA composite, an Oracle Mediator component can provide routing and transformation, a BPEL component can provide business process orchestration, a human task service can provide workflows, and a decision service can provide complex business rules based decision making.

**Design Pattern Summary:** A business component in the ADF Business Components Framework publishes a business event to execute a SOA composite application. The SOA composite application subscribes to the event using the Oracle Mediator component, and from there it can be routed to any other service component, such as BPEL.

**Involved components:**
- Oracle ADF web application that includes ADF Business Components.
- SOA composite application that includes an Oracle Mediator service component and an additional service component to which the event can be routed (such as a BPEL process service component).

**34.1 Introduction to the Recommended Design Pattern**

Oracle Fusion applications initiate business processes in response to user actions. Oracle ADF provides a change notification framework that is triggered at the end of a transaction involving ADF Business Components. This notification can be declaratively configured to raise business events that conform to an Event Description Language (EDL) definition. When an event is raised, it is published on the Event Delivery Network (EDN).

Process flows are implemented using a BPEL process service component as a bridge from EDN to the BPEL process. Because the events raised by the ADF Business Components are not a native BPEL construct, you use a mediator service component to subscribe to the event and to then invoke the BPEL process service component. The mediator service component acts as a binding between the EDN and the BPEL process service component. Whenever the event is raised by ADF Business Components (whether through a GUI action or programatically), the BPEL process service component is invoked. **Figure 34-1** illustrates how these work together.
This approach is recommended for the following reasons:

- Validation logic can be consolidated in the entity object.
- Multiple user interfaces can invoke the reusable entity object that produces the events.
- Event subscription can be modeled on the mediator service component and not hard wired into the UI.
- The producer of the event (in this case, ADF Business Components) does not need to know who the downstream consumers of the event are. If needed, the SOA back-end service can change, without needing to change anything in the Oracle ADF web application. Decoupling the Oracle ADF and SOA application lifecycles makes development more manageable.

Events raised by ADF Business Components are asynchronous with no return value. The event infrastructure leverages the WLS JMS provider, so any unconsumed events can be de-queued by the SOA platform at some later time if the platform isn’t running, assuming the JMS implementation leverages Oracle Advanced Queuing.

**Related Links**

The following documents provide additional information related to subjects discussed in this section:

- For more information about the EDN, see the chapter “Using Business Events and the Event Delivery Network” in the *Developing SOA Applications with Oracle SOA Suite.*

- For information about integrating Oracle Advanced Queuing with Oracle BPEL Process Manager or Oracle Mediator, see the chapter “Oracle JCA Adapter for AQ” in the *Understanding Technology Adapters.*
34.2 Other Approaches

Instead of using ADF Business Components, and the change notification publisher in entity objects to invoke a BPEL service component, you could use one of the following approaches. Use these development approaches only when the recommended approach cannot be implemented.

- **Using the Java Event API to Publish Events**
- **Using a JAX-WS Proxy to Invoke a Synchronous BPEL Process**

**WARNING:**

The following approaches should not be used:

- A web services data control created using a SOAP web service on the composite to create the view in the web application.
- A custom Java class in the web application using SOAP.
- A direct API call, or some other means to access the SOA composite.

34.3 Example

A web application built using ADF Business Components and Oracle ADF Faces allows users to register bugs found in software. An ADF Business Components entity object is used to create a bug, and contains an event whose payload is the attribute values for the created bug. The event is configured to be raised whenever the Create operation is called on the entity object.

A mediator service component subscribes to the event and accepts the event payload. A routing rule is configured for the mediator service component that routes the payload for the event to a BPEL process service component. This component then sends an email that contains the information from the payload to the bug's creator.

There are some cases in which one might need to propagate the end user ID of the event raiser across the invoked services for auditing purposes. It is recommended to propagate this information in the event payload. When raising events for CUD operations (create, update, delete), include the `last_updated_by` history column in the event definition. As this column exists in every Oracle Fusion Applications table, the user raising the event will always be propagated.

The sample code for this use case can be downloaded from Oracle SOA Suite samples.

34.4 How to Initiate a BPEL Process Service Component from an Oracle ADF Web Application

To initiate a BPEL process service component from a web application, you first need to create the web application using ADF Business Components and Oracle ADF Faces. You then create a SOA composite application that contains a mediator service component to pass the event payload created by ADF Business Components, and execute a BPEL process service component.
To invoke a BPEL process service component from an Oracle ADF web application:

1. In the Oracle ADF web application, define an event on an entity object. When you create the event, ensure the following:
   - Include attributes as needed for the payload.
   - Set the event point to be the database operation (create, update, delete) that will raise the event. You can also create any conditions needed to determine when the event should be raised.

   **Note:**
   - Event points can only be associated with Data Manipulation Language (DML) operations.
   - Personally identifiable information (PII) is any piece of information that can be used to uniquely identify a person. PII is sensitive and must be protected from potential misuse.

   When data is included in events or a BPEL flow, it is potentially exposed. While the transport may be encrypted on the SOA side, the data is not. The data in events, payload and BPEL variables is not secured by the security restrictions for business objects. Consider what data is to be exposed in the payload so as to prevent unauthorized access.

   Defining an event generates an Event Definition Language (.edl) file and XML schema definition (.xsd). The EDL file contains all event definitions for the entity and the XSD file defines the contents of an event’s payload, in addition to other objects needed by the BPEL process service component. These files together define the contract between the Oracle ADF application and the SOA composite application, as for a particular event, they identify the elements the SOA composite expects. Both these files are placed in the *events* directory for the project, and can be found in the Application Navigator as children to the associated entity object.

   In the example bug application, the BugReport entity object contains the BugCreated event. This event carries all the attributes on the entity object as its payload, and is published using the create operation as its event point.

2. Create the page in the view that will invoke the operation defined as the event point for the event (for example, through a command button bound to the commit operation or through the implicit call to the commit operation as a task flow return activity).

   In the example bug application, this is a UI command component bound to the Commit operation on the BugReport entity object. Because this operation commits the data to the database, and the Commit operation’s corresponding DML operation (create) is used to sync the ADF Business Components cache with the database, the ADF Business Components framework raises the event.

3. For verification purposes, you can either run the Oracle ADF web application from the Integrated Oracle WebLogic Server container included with Oracle JDeveloper, or you can deploy the Oracle ADF web application to a standalone container. This step includes procedures for running the Oracle ADF web
application within the embedded container and the SOA composite on a standalone Oracle WebLogic Server container. See Step 4 for procedures on deploying to a standalone container.

a. Make sure that EDN data sources have been configured. Using Oracle WebLogic Server Administration Console, verify that EDNDataSource and EDNLocalTxDataSource have been configured.

Note:
Oracle ADF and SOA data sources for EDN must point to the same schema. The EDN schema cannot be shared by more than one SOA runtime environment (such as outside a cluster).

b. Navigate to Domain Configurations > Services > JDBC > Data Sources to verify the existence of EDN data sources.

c. If the EDN data sources have not been configured, create new EDN data sources. Select EDNDataSource and click New. Enter the following details:
Name: EDNDataSource
JNDI Name: jdbc/EDNDataSource
Database Type: Oracle and Database Driver > Oracle Thin Driver XA: Versions 9.0.1.9.2.0.10.11.
Driver Class Name: oracle.jdbc.xa.client.OracleXADatasource.
Click Next. In the next window, uncheck Supports Global Transactions.
Click Next and configure the following:
Database Name: DB_NAME_FUSION_EDN
Host Name/Port: Enter the host name and port for server running the FUSION_EDN database
Database User Name/Password: Enter a username and password.
Test the data source. Set as DefaultServer and click Finish.

Define EDNLocalTxDataSource as above, but use EDNLocalTxDataSource for the data source and jdbc/EDNLocalTxDataSource for the JNDI name.

d. Map the data source in the web.xml and weblogic.xml files associated with the event publishing application. Add the lines shown in Example 34-1 to the WEB-INF/web.xml file.

e. Create an XML file called weblogic.xml with the following contents, and save it in the WEB-INF directory. This maps the global JMS resources to local JNDI names. The names in this file are the defaults expected by the remote connection factory, so you do not need to specify them. An example is shown in Example 34-2.

4. Add event-related SOA runtime libraries, specifically ADF Business Components uses Event Publishing APIs bundled in fabric-runtime.jar. Add a library reference to oracle.soa.workflow.wc to include the event publishing APIs bundled in the relevant JAR files.
Add the code shown in Example 34-3 to the weblogic-application.xml file.
5. In Oracle JDeveloper, create a SOA composite application project and add a mediator service component.

When you create the project, ensure the following:

- You choose to create a mediator service component on completion.

- Before configuring the mediator service component, manually copy the EDL and XSD files created in Step 1 to the SOA composite application project's source path.

- Open the MPLAN file for the mediator service component and create a new subscription that points to the EDL file moved into the source path. This means the mediator service component will now be subscribed to that event.

In the following sample application, a mediator service component named BugCreatedRouter is subscribed to the BugCreated event, as shown in Figure 34-2.

**Figure 34-2  Mediator Subscription to an Event**

6. Create the BPEL process service component.

When you create the component, ensure the following:

- Use the Create BPEL Process dialog to configure the payload. Use the Input Element finder icon to select the payload from the schema created for the event.

In the sample application, the input element would be the BugCreatedInfo payload under the BugReport.xsd node, as shown in Figure 34-3.
You create an activity that accepts the payload from the input element as its input parameters. In the sample application, an email activity is created that takes various input parameters from the payload as assigns them to the email’s parameters.

7. In the mediator service component, add a routing rule and configure it so that it invokes the BPEL process service and contains a subscription to the ADF Business Components event. Ensure the following:

- You select the initiate operation on the client of the BPEL process service component as the target service, as shown in Figure 34-4.
8. Optionally, use the Transformation map to map the mediator service component’s schema to the input schema of the BPEL process service component. However, this should not be necessary, as you should be using the same schema for both.

9. Deploy the SOA composite application to the SOA infrastructure.

10. Use Oracle Enterprise Manager Fusion Middleware Control Console to view the SOA composite application to ensure it was properly deployed.

Related Links
The following documents provide additional information related to subjects discussed in this section:

- For more information on creating events on entity objects, see the chapter "Creating a Business Domain Layer Using Entity Objects" in the Developing Fusion Web Applications with Oracle Application Development Framework.

- For more information about using business events, see the chapter "Using Business Events and the Event Delivery Network" in the Developing SOA Applications with Oracle SOA Suite.

- For more information about creating views, see "Part IV: Creating a Databound Web User Interface" in Developing Fusion Web Applications with Oracle Application Development Framework.

- For detailed procedures on creating SOA composite application projects, see the chapters "Developing SOA Composite Applications with Oracle SOA Suite" and "Getting Started with Oracle Mediator" in the Developing SOA Applications with Oracle SOA Suite.

- For more information about creating a BPEL process service component, see the chapter "Getting Started with Oracle BPEL Process Manager" in the Developing SOA Applications with Oracle SOA Suite.
• For more information about deploying SOA composites and using Oracle Enterprise Manager Fusion Middleware Control Console, see the chapter "Deploying SOA Composite Applications" in the Developing SOA Applications with Oracle SOA Suite.

Example 34-1  Editing the web.xml File

```xml
<resource-ref>
  <res-ref-name>jdbc/EDNDataSource</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

Example 34-2  Mapping Global JMS Resources to Local JNDI Names

```xml
<?xml version="1.0"?>
<resource-description>
  <res-ref-name>jdbc/EDNLocalTxDataSource</res-ref-name>
  <jndi-name>jdbc/EDNLocalTxDataSource</jndi-name>
</resource-description>
<resource-description>
  <res-ref-name>jdbc/EDNDataSource</res-ref-name>
  <jndi-name>jdbc/EDNDataSource</jndi-name>
</resource-description>
```

Example 34-3  Add a Reference to oracle.soa.workflow.wc to the weblogic-application.xml File

```xml
<library-ref>
  <library-name>oracle.soa.workflow.wc</library-name>
</library-ref>
```

### 34.5 Alternative Approaches

Following are alternative approaches to the use case pattern:

- **Using the Java Event API to Publish Events**
- **Using a JAX-WS Proxy to Invoke a Synchronous BPEL Process**

#### 34.5.1 Using the Java Event API to Publish Events

You can programatically raise events following an ADF Business Components CUD operation using the publishEvent API.

**Before you begin:**

1. In Oracle WebLogic Server Console, set up EDN data sources as described in How to Initiate a BPEL Process Service Component from an Web Application.
2. Add a library reference to oracle.soa.workflow.wc.

**To publish events using the Java event API:**

1. Import the required libraries into your application as shown in Example 34-4.
2. Publish events as required. An example is shown in Example 34-5.
Example 34-4  Importing Files into Your Application

import javax.xml.namespace.QName;
import oracle.fabric.blocks.event.BusinessEventConnection;
import oracle.fabric.blocks.event.BusinessEventConnectionFactory;
import oracle.fabric.common.BusinessEvent;
import oracle.integration.platform.blocks.event.BusinessEventBuilder;
import oracle.integration.platform.blocks.event.BusinessEventConnectionFactorySupport;
import oracle.xml.parser.v2.XMLDocument;
import org.w3c.dom.Element;
import oracle.jbo.server.TransactionEvent;
import oracle.jbo.server.JTATransactionHandler;

Example 34-5  Publishing Events Using the Java Event API

private final String eventName = "CreateExpenseReport";
private final String eventNamespace = "/oracle/apps/ta/model/events/edl/ExpenseReportEO";
private final String schemaNamespace = "/oracle/apps/ta/model/events/schema/ExpenseReportEO";

private BusinessEventConnectionFactory cf = null;
private BusinessEventConnection conn = null;

public void eventSetup()
{
    // Get event connection. Set to true for debugging only.
    BusinessEventConnectionFactory cf =
        BusinessEventConnectionFactorySupport.findRelevantBusinessEventConnectionFactory(false);
    BusinessEventConnection conn = cf.createBusinessEventConnection();
}

private XMLDocument getXMLPayload() {
    Element masterElem, childElem1, childElem2;
    XMLDocument document = new XMLDocument();
    masterElem = document.createElementNS(schemaNamespace, "CreateExpenseReportInfo");
    document.appendChild(masterElem);
    childElem1 = document.createElementNS(schemaNamespace, "Id");
    masterElem.appendChild(childElem1);
    childElem2 = document.createElementNS(schemaNamespace, "newValue");
    childElem2.setAttribute("value", ((BigDecimal) this.getAttribute("Id")).toString());
    childElem1.appendChild(childElem2);
    return document;
}

public void beforeCommit(BusinessEventConnection conn, TransactionEvent e) {
    // Determine whether this is a JTA transaction.
    if (this.getDBTransaction() != null &&
        this.getDBTransaction().getTransactionHandler() instanceof
        JTATransactionHandler) {
        // Determine whether the row is newly created.
        if (this.getEntityState() == STATUS_NEW) {
            try {
                // Build the event.
                BusinessEventBuilder builder = BusinessEventBuilder.newInstance();
                // Specify the event name and namespace. In this example,
they are constants: eventNamespace and eventName.

builder.setEventName(new QName(eventNamespace, eventName));

// Specify the event payload. In this example, the custom
// method getXMLPayload constructs the payload.
builder.setBody(getXMLPayload().getDocumentElement());
BusinessEvent event = builder.createEvent();

// Publish the event.
conn.publishEvent(event, 5);
conn.close();

// For debugging and testing purposes, print the result.
System.out.println("Event was sent sucessfully");
} catch (Exception exp) {
    System.out.println("Failed sending event: " + exp.getMessage());
    exp.printStackTrace();
}
}

super.beforeCommit(e);

34.5.2 Using a JAX-WS Proxy to Invoke a Synchronous BPEL Process

Another way to invoke a SOA composite as a web service from an Oracle ADF web
application is to use a JAX-WS proxy.

Use this pattern only for synchronously invoking BPEL processes where the calling
application waits for a response. As such, any BPEL processes called using this pattern
must be synchronous and brief so as to avoid any time out issues.

---

Caution:

- Do not use this pattern for a long running or asynchronous BPEL process.
- Make sure that synchronous services return immediately. Synchronous
  services should be simple input/output payloads.
- Do not use multi-record or N record services in which processing time
  varies from seconds to minutes or longer.

---

In this pattern, SOA composite services are exposed as web services. You generate a
JAX-WS proxy client to invoke the SOA composite exposed as a web service from your
ADF Business Components application module.

You use the web services wizard to generate a web service proxy, and then call the
web service using method calls.

An indirection for the web service proxy node enables retrieving the location of the
web service WSDL, username and password from connections.xml. To use the
indirection, access the proxy via
oracle.adf.model.connections.webservice.WebServiceConnection.

An example is shown in Example 34-6.
Use the username or SAML token policies for identity propagation and security.

**Related Links**
The following documents provide additional information related to subjects discussed in this section:

- For more information about this approach, see the chapter "Integrating Web Services Into a Fusion Web Application" in *Developing Fusion Web Applications with Oracle Application Development Framework*.

- For more information about securing the use case, see *Securing Web Services Use Cases*.

**Example 34-6  Using WebServiceConnection**

```java
WebServiceConnection wsc = (WebServiceConnection)ADFContext.getCurrent().getConnectionsContext().lookup(connectonName);
Hello hello = wsc.getJaxWSPort(Hello.class);
```

### 34.5.3 Invoking Third-Party Web Services in the Cloud

The typical design pattern described here works for developers and implementers who subscribe to a business event and invoke third-party services in BPEL via partner links. However, when using Software as a Service (SaaS), it is not always possible to simply deploy composites with custom-built BPEL processes. In these cases, the following recommendations may be useful.

- Invoke a SOA composite with a seeded web service callout to the registered URL with a pre-determined WSDL file one way notification. You can implement this pattern using Oracle Java Cloud Service.

- Invoke a SOA composite with a seeded w service callout to a registered URL with a pre-determined WSDL file request response. You can implement this pattern using Oracle Java Cloud Service.

- Invoke SOA composite with a seeded web service callout to a registered Oracle Business Process Management process. You can implement this pattern using Oracle Business Process Management Composer and then create the third-party service using Oracle Business Process Management Business Catalog.

### 34.6 Secure the Design Pattern

To secure this pattern, it is recommended that you secure the Oracle ADF web application. For more information about securing the pattern, see *Securing Web Services Use Cases*.

#### 34.6.1 Run the Mediator as an Event Publisher

To make the mediator run as an event publisher:

1. Open the composite.xml file for the SOA composite application and manually add the `runAsRoles="$publisher"` attribute to the composite subscriptions. 

   Example 34-7 shows the composite subscription for the sample application.
2. To validate the subject propagation in the SOA composite application, add the following code as shown in Example 34-8 to the BPEL file for the BPEL process flow:

**Example 34-7  runAsRoles Attribute in a Composite Subscription**

```xml
<component name="BugReportMediator">
  <implementation.mediator src="BugReportMediator.mplan" />
  <business-events>
    <subscribe xmlns:sub1="/model/events/edl/BugReport"
              name="sub1:bugCreated"
              consistency="oneAndOnlyOne"
              runAsRoles="$publisher"/>
  </business-events>
</component>
```

**Example 34-8  Using bplex:exec to Print Out Subject Information**

```xml
<bpelix:exec language="java" version="1.5">
  <![CDATA[
    javax.security.auth.Subject
    subject = javax.security.auth.Subject.getSubject
    ( java.security.AccessController.getContext());
    System.out.println("\n\n\n\n\n\n");
    System.out.println("######*****----->
                     subject:  " + subject.toString());
    System.out.println("\n\n\n\n\n\n");
  ]]>}
</bpelix:exec>
```

### 34.6.2 Securing Event-Driven Applications

Events enable event-driven applications and are not related to OWSM. Therefore, OWSM policies do not apply to events.

Events raised by Oracle ADF web applications automatically propagate the event's publisher ID in the event header. No action is required to perform identity propagation. The publisher's ID corresponds to the end-user authenticated in the application.

### 34.7 Verify the Deployment

You can verify the deployment by testing the Oracle ADF web application. Alternatively, you can send EDN events at the command line to verify event-raising functionality.

### 34.7.1 How to Verify the Deployment

To properly verify this design pattern, you should test the Oracle ADF web application and enable logging, use Oracle Enterprise Manager Fusion Middleware Control Console to verify the process instance creation, and check the SOA logs.
To verify this design pattern:

1. Test your Oracle ADF web application. Be sure to enable ADF Business
Components runtime logging by setting the Java VM parameter in the run/debug
profile to \`jbo.debugoutput=console.\` Doing so logs the event and its payload
in the JDeveloper log console.

2. Run the Oracle ADF web application and invoke the method that raises the event.

3. View the ADF Business Components runtime log in the JDeveloper log console to
view the event and payload.

4. Use Fusion Middleware Control Console for tracking composite instances and for a
variety of debugging, monitoring and testing functions.

   Launch Fusion Middleware Control Console using the following URL:

   \http://<hostname>:<port number>/em\n
5. Open Fusion Middleware Control Console to verify the process instance has been
created. Use the Audit tab to verify that the payload is correct.

34.7.2 How to Test EDN Functionality from the Command Line

You can test EDN functionality using SendEvent and edn.debug.event-
connection at the command line.

34.7.2.1 SendEvent

SendEvent is a command line utility for sending an event to a database-based Oracle
Event Delivery Network instance. SendEvent can send an empty event (of a given
queue name) or an entire event from a file. Running the command displays a list of
command line options.

Some examples are shown in Example 34-9 and Example 34-10.

In Example 34-9, an empty event with namespace \`uuid:1111\` and local name
MyEvent is sent to the event bus.

In Example 34-10, the event contained in the file \`AnEvent.xml\` is sent to EDN.

**Example 34-9  Sending an Empty Event**

```bash
oracle.integration.platform.blocks.event.SendEvent -dbconn host.example.com:1521:SID
-dbuser user -dbpass password -eventName {uuid:1111}MyEvent
```

**Example 34-10  Sending an Event to EDN**

```bash
oracle.integration.platform.blocks.event.SendEvent -dbconn host.example.com:1521:SID
-dbuser user -dbpass password -event AnEvent.xml
```

34.7.2.2 BusinessEventConnectionFactorySupport

You can use BusinessEventConnectionFactorySupport in your Oracle ADF
web application to test your event publishing code. Rather than sending an event to a
queue, you can configure your Oracle ADF web application to print the event
information to the log using BusinessEventConnectionFactorySupport.

To do so, set the system property `edn.debug.event-connection` to `true` when
running your application. When the application sends an event, the information for
that event is logged, including the event body in its entirety. This enables you to see
the events that will be sent to EDN when the application runs on a SOA server.
The log name is `oracle.integration.platform.blocks.event.debug`, but in most configurations the information prints to `stdout` by default.

**To set the system property:**

When Oracle WebLogic Server starts up, add the system property shown in Example 34-11 to the `JAVA_OPTIONS` environment variable.

**To access the EDN database log:**

You can use the EDN database log shown in Example 34-12 for diagnostic purposes.

**Example 34-11 Setting the EDN Debug System Property**

```java
-Dedn.debug.event-connection=true
```

**Example 34-12 EDN Database Log**

```
http://HOSTNAME:7001/soa-infra/events/edn-db-log
```

### 34.7.3 Related Links

The following documents provide additional information related to subjects discussed in this section:

- For more information about testing and debugging Oracle ADF web applications, see the chapter "Testing and Debugging ADF Components" in the *Developing Fusion Web Applications with Oracle Application Development Framework*.

### 34.8 Troubleshoot the Use Case

Following are tips that may help resolve common issues that arise when developing or running this use case.

#### 34.8.1 Deployment

If deployment of the SOA composite application fails, then verify the following:

- The `location` element in the EDL file located in the directory is relative to the directory that contains the EDL file.
- If you get an invalid XPath expression exception, use a static value instead of a value from the payload.

#### 34.8.2 Runtime Errors

If your Oracle ADF web application encounters a runtime class load error for `EventConnectionFactory`, you need to add a library reference to `oracle.soa.workflow.wc`.

### 34.9 What You May Need to Know About Initiating a SOA Composite from an Oracle ADF Web Application

Before you implement these design patterns, you should be aware of the following:

- If you change the event name or event payload, the mediator will not respond to the raising of the event. If you need to make changes, you should create a new event. During development, you need to change the mediator (if the name of the event changes) and/or the BPEL process being invoked (if the payload changes).
34.10 Known Issues and Workarounds

Following are known issues:

- You must add a library reference to `oracle.soa.workflow.wc` to raise events successfully.
- You can only specify `$publisher` as the `runAsRole` for an event subscription.
Initiate a SOA Composite from a PL/SQL Stored Procedure

This chapter describes what a PL/SQL stored procedure needs to do to initiate a SOA composite application.

**When to implement:** When a PL/SQL stored procedure needs to initiate a SOA composite application.

**Design Pattern Summary:** A PL/SQL stored procedure raises an event through the Event Delivery Network within the database. A mediator in the SOA composite application subscribes to the event and routes it as appropriate.

**Involved components:**
- PL/SQL stored procedures
- Event Delivery Network database
- SOA composite application that includes Oracle Mediator and other components as needed.

### 35.1 Introduction to the Recommended Design Pattern

Oracle Fusion applications may contain stored procedures that need to invoke a component within a SOA composite application, such as a BPEL process service component. A stored procedure can use the Event Delivery Network database API to publish an event whose payload is `xmltype`. An Oracle Mediator service component subscribes to the event by event name or by using a XPath expression on the event payload. The .edl file (*event definition file*) for the event can be supplied in the composite or deployed separately in a MAR (*metadata archive*). When the stored procedure publishes the event, the subscribed Oracle Mediator service component forwards the payload to the BPEL process service component.

This chapter explains how to implement the recommended approach.

### 35.2 Example

The sample code for this use case can be downloaded from Oracle SOA Suite samples.

### 35.3 How to Invoke a SOA Composite Application Component Using PL/SQL

To invoke a SOA composite application component from a stored procedure, you must first create the event within the SOA composite application. The stored procedure must then raise the event and pass any required data via the EDN database API.
To invoke a SOA composite application component using PL/SQL:

1. Create a SOA composite application with an Oracle Mediator component.

2. Configure Oracle Mediator to subscribe to a new event (with a name of your choosing).

   The event filter can be by event name or using an XPath expression on the event payload and the EDL for the event can either be supplied in the composite.xml or deployed separately in a MAR.

3. Create the SOA composite application component that will be invoked (for example, a BPEL process service component), and create a wire between the Oracle Mediator component reference and the component service.

4. From a PL/SQL stored procedure, call the EDN-DB API method publish_event with the event namespace and the event payload as a CLOB type. An example is shown in Example 35-1.

Example 35-1  Calling the publish_event Method

```sql
DECLARE
  NAMESPACE VARCHAR2(200);
  LOCAL_NAME VARCHAR2(200);
  PAYLOAD CLOB;
BEGIN
  NAMESPACE := 'http://xmlns.oracle.com/SubEventMediator/EventDefinition1';
  LOCAL_NAME := 'CustomerEvent';
  PAYLOAD := to_clob('<eb:business-event xmlns:eb=
xmlns:ob="http://xmlns.oracle.com/SubEventMediator/EventDefinition1">
  <eb:name>ob:CustomerEvent</eb:name><eb:content><CU:CustomerData
xmlns:CU="http://xmlns.oracle.com/Esb/CustomerData
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <CustomerId>A22-9AXC2</CustomerId><CustomerName>Deserae International</CustomerName><Type>Gold</Type><Description>Accounting Outsourcing Partner</Description><Address>3228 Massilon Blvd</Address><City>Juniper</City><State>Massachusetts</State><Zip>01854</Zip><Country>US</Country><Phone>877-555-9876</Phone><Status>Active</Status><CreditRating>5</CreditRating><Discount>0</Discount><Terms>30n4</Terms><EnrollDate>01/1/01</EnrollDate><LastOrderDate>05/05/05</LastOrderDate><Currency>USD</Currency><ContactName>Jan Forester</ContactName><ContactTitle>VP Finance</ContactTitle><ContactPhone>877-555-9000</ContactPhone><AccountRep>Geoff Seattle</AccountRep><CampaignRating>2</CampaignRating><ReferredBy>Houston America Taxco</ReferredBy><ContactName></CU:CustomerData></eb:content></eb:business-event>);
  EDN_PUBLISH_EVENT( NAMESPACE => NAMESPACE, LOCAL_NAME => LOCAL_NAME, PAYLOAD => PAYLOAD);
END;
```

35.4 Securing the Design Pattern

Secure Oracle Mediator by configuring the property runAsRoles=$publisher.

When the database connection is established from the middle tier so as to invoke the PL/SQL stored procedure, a session is established with the appropriate identity. This identity is propagated through EDN back to the middle tier for the subscription. The subscription runs as the identity of the publisher.
Related Links
The following documents provide additional information related to subjects discussed in this section:

- For details on securing the Oracle Mediator, see Securing the Design Pattern.
- For more information about securing this pattern, see Securing Web Services Use Cases.

35.5 Verify the Deployment
Verifying the deployment involves the following:

- Testing and Deploying the Use Case
- Verifying the SOA Composite Deployment Using Oracle Enterprise Manager Fusion Middleware Control Console

35.5.1 Test and Deploy the Use Case
Testing and deploying the use case involves the following main steps:

1. Test your Oracle ADF application.

2. Deploy the SOA composite application to the standalone WLS where the SOA infrastructure has been installed. Because you created a published event from the SOA composite application to the ADF Business Components service, the ADF Business Components service need not to also be deployed to the SOA infrastructure.

3. Test the deployed SOA composite service using Oracle Enterprise Manager Fusion Middleware Control Console. Every deployed service has its own test page, so you can quickly test that the service functions as you expect.

35.5.2 Verifying the SOA Composite Deployment Using Oracle Enterprise Manager Fusion Middleware Control Console
You can use Oracle Enterprise Manager Fusion Middleware Control Console to verify that the SOA composite was successfully deployed. In Oracle Enterprise Manager Fusion Middleware Control Console, you can select the SOA composite instance and display the result of the event.

Using Oracle Enterprise Manager Fusion Middleware Control Console, you can:

- Verify the deployment of the SOA composite.
- Test the SOA composite.
- Verify the SOA composite test results.

To verify that the SOA composite was successfully deployed and the event was received:

1. Using a web browser, access the Oracle Enterprise Manager Fusion Middleware Control Console using a URL such as the following:

   http://<host name>:<port number>/em
2. From the list of applications, expand the PLSQLEvent composite.

3. In the Last 5 Instances pane, click the most recent instance as shown in Figure 35-1.

Figure 35-1  Finding the Latest PLSQLEvent Composite Instance

4. In the Flow Trace window that displays, click the Oracle Mediator component, as shown in Figure 35-2.

Figure 35-2  The Flow Trace Window

A window displays, showing the event results, as shown in Figure 35-3.
35.5.3 Check the Output Log File to Verify that the Event was Received

You can check the output log file to verify that the Mediator received the event.

To verify that the event was received using the output log file:

1. Navigate to the `/tmp/output` directory.

2. Open the `p_n.xml` file, where `n` is a sequential number indicating the application instance. For example, running the application twice results in two files under the `/tmp/output` directory: `p_1.xml` and `p_2.xml`.

3. Examine the file in order to verify that the event was received.

For example, the payload log file in Example 35-2 indicates that the event was received (as shown in the bold text):

Example 35-2  Example p_n.xml file

```
2007-07-25 13:16:02.062 NOTIFICATION Immediate case {WriteCustomer.Write} will be executed
2007-07-25 13:16:02.070 NOTIFICATION No Condition defined
2007-07-25 13:16:02.131 NOTIFICATION Transforming the source message to target
2007-07-25 13:16:02.132 NOTIFICATION Source message properties
    :{tracking.compositeInstanceId=151,
     tracking.ecid=140.87.4.171:75256:1185394211665:0:1774896848,
     id=d76fe830-cbdf-4338-8a1a-4687ff463a1a}
2007-07-25 13:16:02.167 NOTIFICATION Getting part {payload} from message:in
2007-07-25 13:16:02.171 NOTIFICATION Calling the real transformer
2007-07-25 13:16:02.174 NOTIFICATION setting out payload to part :CustomerData
2007-07-25 13:16:02.181 NOTIFICATION Source payload =
    ...
2007-07-25 13:16:02.184 NOTIFICATION Transformed payload =
    <cust:CustomerData xmlns:cust="http://xmlns.oracle.com/Esb/CustomerData">
    ...
2007-07-25 13:16:02.185 NOTIFICATION getPayload: normal payload
2007-07-25 13:16:02.252 NOTIFICATION DEBUG: Mediator sending a request to reference WriteCustomer to its operation = Write
```

Initiate a SOA Composite from a PL/SQL Stored Procedure  35-5
35.5.4 Related Links

The following documents provide additional information related to subjects discussed in this section:

- For more information about testing and debugging your Oracle ADF application, see the chapter "Testing and Debugging ADF Components" of the Developing Fusion Web Applications with Oracle Application Development Framework.

- For information about testing the ADF Business Components service, see the chapter "Integrating Web Services Into a Fusion Web Application" in the Developing Fusion Web Applications with Oracle Application Development Framework.

- For more information about using the Fusion Middleware Control Console to test deployed SOA composite applications, see the chapter "Automating Testing of SOA Composite Applications" in the Developing SOA Applications with Oracle SOA Suite.

35.6 Troubleshooting the Use Case

Following are tips that may help resolve common issues that arise when developing or running this use case.

- Enable logging for Oracle Mediator using logging.xml.

- For the events functionality, use the Event Delivery Network database log page at http://host:port/soa-infra/events/edn-db-log. The EDN schema name is FUSION_EDN.

Related Links

See the troubleshooting section in the chapter "Deploying SOA Composite Applications" of the Oracle SOA Suite Developer’s Guide for more information.

35.7 What You May Need to Know About Initiating a SOA Composite from a PL/SQL Stored Procedure

Before you implement these design patterns, be aware of the following:
Run the sample provided before implementing your own version of this use case. Running the sample ensures that the EDN database queue works as expected.

### 35.8 Known Issues and Workarounds

Following are known issues:

- Event publishing is an asynchronous action, there is no support for synchronous event publishing.
Orchestrate ADF Business Components Services

This chapter describes how to use a SOA composite application to invoke business methods within an Oracle ADF web application. In this pattern, the Oracle ADF web application business methods make changes to data, whereas the SOA composite does not.

**When to implement:** This pattern describes how to use a SOA composite application to invoke business methods within an Oracle ADF web application. In this pattern, the Oracle ADF web application business methods make changes to data, whereas the SOA composite does not. For example:

- A BPEL process service component must retrieve data from a database using an ADF Business Components service. However, the BPEL process service component will not post any changes back to the database.

- A BPEL process service component must access an exposed service method on an ADF Business Components service, and that method contains only business logic and does not update data.

- A BPEL process service component must access an exposed service method on an ADF Business Components service. The exposed service method on the business component service does not require a conversational callback style of interaction. Instead, it provides a single invocation, wrapping all of the business logic and view object manipulation. An example of this is a service method that deletes an order and all line items, given an order ID as a parameter.

**Related Links**

The following document provides additional information related to subjects discussed in this section:

See [Manipulating Back-End Data from a SOA Composite](#) for information regarding patterns in which both the Oracle ADF web application and SOA composite must update data without conflicting.

**Design Pattern Summary:** A BPEL process service component uses an invoke activity to invoke a partner link that accesses a SOAP service created for a business component.

**Involved components:**

- SOA composite application that includes a BPEL process service component
- Oracle ADF web application that includes ADF Business Components
36.1 Introduction to the Recommended Design Pattern

Oracle Fusion applications may require that a BPEL process service component access data values from ADF Business Components in an Oracle ADF web application. Oracle Fusion applications may also require that a BPEL process service component invoke a method contained in ADF Business Components. Instead of directly accessing the ADF Business Components, you can publish the component as a web service. The composite then accesses the published component over SOAP using an invoke activity and partner link in the BPEL process service component. Figure 36-1 shows a high-level overview of this design pattern.

![Figure 36-1 SOA Composite Application Accesses ADF Business Components Using SOAP](image)

This approach is recommended because SOAP bindings do not require that the Oracle ADF web application and the SOA components be co-located in the same container.

36.2 Other Approaches

There are no other approaches to implementing this use case. The only supported way to invoke ADF Business Components services is to use a web service SOAP binding.

**WARNING:**

Using the Oracle ADF binding to invoke ADF Business Components services is prohibited due to topology and security requirements. (Oracle ADF services and SOA composites must be collocated; there are no application roles and privileges cannot be escalated.)

36.3 Example

The sample code for this use case can be downloaded from *Oracle SOA Suite samples*.
36.4 How to Invoke an ADF Business Components Service from a BPEL Process Service Component

To invoke ADF Business Components using a SOAP binding, you first publish the business component as a web service. You then create a BPEL process service component that contains a partner link to the ADF Business Components service. The partner link is accessed from an invoke activity. An assign activity is used to populate data into a variable used to pass data to the ADF Business Components service.

To invoke ADF Business Components from a BPEL process service component:

1. Create ADF Business Components, including an application module.

2. Optionally, create any SDO classes for view objects.
   Any top-level view objects referenced in a service interface created from an application module will automatically be service-enabled. However, creating the SDO classes for individual view and entity objects allows you to configure the SDO name or namespace, or selectively service-enable child view objects.

3. Configure the service interface for the application module.
   You can elect to include in your service interface custom methods, top-level service view instances, and any find operations based on view criteria.
   At the end of this step, Oracle JDeveloper creates the WSDL files that will be used by the BPEL process service component to access any required methods or data. Figure 36-2 shows the WSDL created for StoreFrontService in the sample application. This service accesses the customer and order information needed by the SOA composite application.

4. Implement the security as described in Securing the Design Pattern.

5. For verification purposes, you can either run the Oracle ADF web application from the Integrated Weblogic Server container included with Oracle JDeveloper, or you can deploy the Oracle ADF web application to a standalone container. To deploy to a standalone container:
   a. Create a deployment profile for the web application.
b. Make an Oracle WebLogic Server connection to the container.

c. Deploy the profile to the container.

d. Add a library reference to oracle.soa.workflow.wc. Add the following entry to the file in MW_HOME/user_projects/domains/<domain name>/config/config.xml directory, and add the line shown in Example 36-1 to oracle.adf.domain.loader.

Note:
Skip this step if deploying the profile to the WLS container included in the SOA installation.

6. Add the WSDL file for the ADF Business Components service to the Resource Palette of Oracle JDeveloper. If you are using the embedded server, you first need to run your application. You then create a new URL connection in the Resource Palette to the embedded server. The URL is:

http://host:7101/ApplicationName-ProjectName-context-root/AppModuleService Name

The URL for the embedded server is generated at the bottom of the WSDL file. Alternatively, you can open the file and copy the soap:address URI.

If you have deployed the Oracle ADF web application to a standalone server, then create a new Application Server connection in the Resource Palette. For more information about using the Resource Palette, see the Oracle JDeveloper Online Help.

7. Create a SOA composite application that includes a BPEL process service component.

8. In the SOA composite application, create the reference web service binding to the WSDL file for the ADF Business Components service. When creating the binding, be sure to select the WSDL from the Resource Palette.

Note:
The ADF Business Components service must be running in order for it to be discoverable.

Figure 36-3 shows the composite.xml file for the sample application.
A Partner Link for the ADF Business Components is automatically created when the reference is wired to the BPEL process.

9. Wire the BPEL component to the Oracle ADF web service
   Use a web service binding so that the ADF Business Components service can be remotely deployed.

10. Create an Invoke activity that invokes the partner link for the ADF Business Components.

11. Create an Assign activity to assign any values (for example, a static XML fragment) to variables for the BPEL process service component. These can then be passed to the ADF Business Components service.

12. Repeat adding assign and invoke activities as needed. Figure 36-4 shows the BPEL flow for the sample application.
13. Implement the security as described in Securing the Design Pattern.

Related Links
The following documents provide additional information related to subjects discussed in this section:

- For more information about creating ADF Business Components, see Part II: Building Your Business Services in the Developing Fusion Web Applications with Oracle Application Development Framework.
• For more information about configuring a service interface for an application module and creating SDO classes for view objects, see the chapter Integrating Service-Enabled Application Modules in the Developing Fusion Web Applications with Oracle Application Development Framework.

• For more information about creating a deployment profile for a web application, see the chapter Deploying Fusion Web Applications of the Developing Fusion Web Applications with Oracle Application Development Framework.

• For more information about creating SOA composite applications, as well as bindings and wiring references, see the chapter Developing SOA Composite Applications with Oracle SOA Suite in the Developing SOA Applications with Oracle SOA Suite.

• For more information about creating Assign and Invoke activities, see the chapter Getting Started with Oracle BPEL Process Manager in the Developing SOA Applications with Oracle SOA Suite.

Example 36-1 Shared Library Name

```xml
<shared-library name="adf.oracle.domain" version="11.1.1" library-compatible="true">
...
</import-shared-library name="oracle.soa.workflow.wc"/>
```

36.5 Securing the Design Pattern

You need to secure the following:

• Secure the ADF Business Components web service data control and the SOA composite using SAML policies.

• Secure the ADF Business Components web service and the SOA composite with username token policies.

For information about securing the design pattern, see Securing Web Services Use Cases.

36.6 Verifying the Deployment

To properly verify this design pattern, you should test your ADF Business Components, then deploy and verify the SOA composite application.

To verify this design pattern:

1. Test your Oracle ADF web application.

2. Deploy the SOA composite application to the standalone WLS where the SOA infrastructure has been installed. Because you created a web service binding from the SOA composite application to the ADF Business Components web service, the ADF Business Components web service need not be deployed to the SOA infrastructure.

3. Test the deployed SOA composite service using Fusion Middleware Control Console. Every deployed service has its own test page, so you can quickly test that the service functions as you expect.

Related Links
The following document provides additional information related to subjects discussed in this section:

- For more information about testing and debugging your Oracle ADF web application, see the chapter *Testing and Debugging ADF Components* of the *Developing Fusion Web Applications with Oracle Application Development Framework*.

- For information about testing the ADF Business Components service, see the chapter *Integrating Service-Enabled Application Modules* chapter of the *Developing Fusion Web Applications with Oracle Application Development Framework*.

- For more information about testing a deployed SOA composite service, see *Managing SOA Composite Application Instances* in the *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

### 36.7 Troubleshooting the Use Case

Following are tips that may help resolve common issues that arise when developing or running this use case.

Use Oracle Enterprise Manager Fusion Middleware Control Console to troubleshoot the use case:

http://localhost:8888/em

You can test your SOA composite using Fusion Middleware Control Console.

**Related Links**

The following document provides additional information related to subjects discussed in this section:

For more information about testing your SOA composite, see the section "Automating Testing for SOA Composite Applications" in the chapter *Managing SOA Composite Application Instances* in the *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

### 36.8 What You May Need to Know About Orchestrating ADF Business Components Services

Before you implement these design patterns, you should be aware of the following:

Any time you invoke an ADF Business Components service, the database transaction for that operation is committed when it completes.

There is no session propagation between the BPEL process and Oracle ADF. While the identity is passed in when security policies are in place, a new Oracle Fusion Data Security session is created with each invocation of the ADF Business Components service operations.

If you invoke ADF Business Components service operations that use event-raising entities, those events are not raised.
This chapter describes what updates created in a SOA composite application need to do to perform create, read, update, or delete (CRUD) operations on back-end data stored in a database.

**When to implement:** When updates created in a SOA composite application need to perform create, read, update, or delete (CRUD) operations on back-end data stored in a database.

**Design Pattern Summary:** Entity variables within a BPEL process service component access ADF Business Components view objects through a service to fetch data on the back end. The BPEL process service component then manipulates the data, and the changes synchronize with the ADF Business Components service when the BPEL service dehydrates. This is also known as master detail with indexing.

An example of master detail with indexing is entity variables created on master detail records such as an order header with lines, accessing the lines individually with array subscripting.

**Involved components:**

- SOA Composite with a BPEL process service component and a SOAP binding.
- ADF Business Components, including view objects, with a published web service interface.

### 37.1 Introduction to the Recommended Design Pattern

When a BPEL process service component needs to perform CRUD operations on back-end data stored in a database, you use BPEL entity variables. Entity variables can fetch data from an ADF Business Components view object through a web service interface, and manipulate the data using common BPEL assign and XPath constructs. These changes automatically synchronize with the ADF Business Components service when the BPEL instance dehydrates. Using entity variables provides the following:

- Ease-of-use by abstracting data manipulation
- Performance gains from the use of SDO change summaries
- Strongly typed XML document to back XML element manipulations

The transaction locking strategy for SDO is optimistic. If the BPEL engine tries to update an SDO or entity variable and the current revision number is out of date, an exception will be thrown by the ADF Business Components service.
37.2 Example

The sample code for this use case can be downloaded from Oracle SOA Suite sample.

37.3 How to Manipulate Data from a BPEL Process Service Component

To manipulate data, first you create an ADF Business Components entity object that accesses and updates the data. You then publish the business component as a web service. Next, you create a SOA composite application that includes a BPEL service component to which you add entity variables that can manipulate the data.

To manipulate data from a BPEL process service component:

1. Create ADF Business Components, including an application module.

2. Optionally create any SDO classes for view objects.

   When modeling your view objects, it is important to determine which may need to be available for binding to SOA composite entities. It is possible to automatically enable services in top-level view objects referenced in a service interface create from an application module. However, you must create the SDO classes for individual view objects so as to configure the SDO name or namespace, or selectively service-enable child view objects.

3. Configure the service interface for the application module.

   You can elect to include in your service interface custom methods, top-level service view instances, and any find operations based on view criteria.

   At the end of this step, JDeveloper creates the WSDL files that will be used by the BPEL process service component to access any required methods or data.

4. Deploy the services to the Integrated Oracle WebLogic Server by right-clicking the nameServiceImpl class and selecting Run. Alternatively, define an ADF Business Components service interface profile and deploy it to the standalone Oracle WebLogic Server.

5. Add the WSDL file for the ADF Business Components service to the Resource Palette. If you are using the integrated server, you must first run your application. In the Resource Palette, create a new URL connection to the integrated server. The URL is:

   http://localhost:7001/ApplicationName-ProjectName-context-root/AppModuleService Name

   The URL for the embedded server is generated at the bottom of the WSDL file. Alternatively, you can open the file and copy the soap:address URI.

   If you have deployed the Oracle ADF web application to a standalone server, then create a new Application Server connection in the Resource Palette. For more information about using the Resource Palette, see JDeveloper Online Help.

**Note:**

The ADF Business Components service must be running in order for it to be discoverable.
6. Create a SOA composite application that includes a BPEL process service component.

7. In the SOA composite application, create a reference binding from the BPEL process to the WSDL file for the ADF Business Components service. When creating the reference binding, be sure to select the WSDL from the Resource Palette.

Note: The ADF Business Components service must be running in order for it to be discoverable.

8. In the BPEL process service component, create an entity variable. This type of variable delegates BPEL data manipulation operations to an underlying data provider implementation, in this case the business component.

When you create the entity variable, ensure that you select the business component as the Partner Link.

9. Add a Bind activity to the BPEL process service component. This activity establishes the key to pass to the ADF Business Components service when the SDO will be fetched from the database.

Example 37-1 shows the XML for a bind activity that establishes OrderId as the key that will be passed to retrieve orders.

10. Add an Assign activity to the BPEL process service component that will manipulate the data.

Example 37-2 shows the XML for an assign activity that first assigns the value from the entity variable to another variable, then takes the value of the entity variable and manipulates it using an XPath expression. The change operations are mapped back to the SDO automatically. For more information about using XPath operations, see Support for XPath Operations.

When the BPEL process service component hits a breakpoint activity (receive, onMessage, wait, onAlarm) the instance is dehydrated. When dehydration happens, or the scope where entity variables are declared completes, all related entity variables that have been loaded flush their changes back to the business component, and from there to the database. For more information, see When Entity Variables Flush Changes Back to ADF Business Components.

You can use sensor variables to monitor the service data object. For more information, see the following links:

11. Deploy the ADF Business Components web service to the Integrated Oracle WebLogic Server with SOA infrastructure. Then deploy the SOA composite application.

Example 37-1 Bind Activity Establishes the Key

```
<bpelx:bindEntity name="BindEntity_1" variable="orderEntityVar">
  <bipelx:key keyname="ns4:OrderId">bpws:getVariableData('inputVariable', 'payload', '/client:BPELEntityVarADFBCProcessRequest/client:orderID')
  </bipelx:key>
</bipelx:bindEntity>
```
Example 37-2  Assign Activity Manipulates an Entity Variable Value

<assign>
  <copy>
    <from expression="$orderEVar/fdsm:OrderTotal + 1" />
    <to variable="orderEVar" query="fdsm:OrderTotal" />
  </copy>
</assign>

Related Links
The following documents provide additional information related to subjects discussed in this section:

- For more information about creating ADF Business Components with an application module, see the chapter "Implementing Business Services with Application Modules" in the Developing Fusion Web Applications with Oracle Application Development Framework.

- For more information about creating SDO classes for view objects and configuring service interfaces for application modules, see the chapter "Integrating Service-Enabled Application Modules" in the Developing Fusion Web Applications with Oracle Application Development Framework.

- For more information about testing and deploying services, see the sections "How to Test the Web Service Using Integrated Oracle WebLogic Server" and "How to Deploy Web Services to Oracle WebLogic Server" in the chapter "Integrating Service-Enabled Application Modules" in the Developing Fusion Web Applications with Oracle Application Development Framework.

- For more information about creating a SOA composite application that includes a BPEL process service component, see the chapter "Developing SOA Composite Applications with Oracle SOA Suite" in the Developing SOA Applications with Oracle SOA Suite.

- For more information about creating entity variables, see the chapter "Manipulating XML Data in a BPEL Process" of the Developing SOA Applications with Oracle SOA Suite.

- For more information about Bind activities, see the section "Adding Service Binding Components" in the chapter "Developing SOA Composite Applications with Oracle SOA Suite" in the Developing SOA Applications with Oracle SOA Suite and "Using ADF Model in a Fusion Web Application" in the Developing Fusion Web Applications with Oracle Application Development Framework.

- For more information about using sensors, see the chapter "Using Oracle BPEL Process Manager Sensors" in the Developing SOA Applications with Oracle SOA Suite.

- For more information about monitoring BPEL process service components, see the chapter "Monitoring BPEL Process Service Components and Engines" in the Administering Oracle SOA Suite and Oracle Business Process Management Suite.

- For more information about managing SOA composites, see the chapter "Managing SOA Composite Application Instances" in the Administering Oracle SOA Suite and Oracle Business Process Management Suite.
37.4 Securing the Design Pattern

To secure this pattern, it is recommended that you follow the same steps as described in Securing the Design Pattern.

37.5 Verifying the Deployment

To properly verify this design pattern, you should test your ADF Business Components service, then deploy and test the SOA composite application.

To verify this design pattern:

1. Test your Oracle ADF web application.

2. Deploy the SOA composite application to the standalone WLS where the SOA infrastructure has been installed. Because you created a web service binding from the SOA composite application to the ADF Business Components web service, the ADF Business Components web service need not be deployed to the SOA infrastructure.

3. Test the deployed SOA composite service using Fusion Middleware Control Console. Every deployed service has its own test page, so you can quickly test that the service functions as you expect.

Tip:

The use of entity variables greatly affects the ability to unit test the process in an isolated environment. Unit tests of processes that use entity variables usually require managing the backend database state.

Related Links

The following documents provide additional information related to subjects discussed in this section:

- For more information about testing and debugging your Oracle ADF web application, see the chapter Testing and Debugging ADF Components of the Developing Fusion Web Applications with Oracle Application Development Framework.

- For information about testing the ADF Business Components service, see the chapter Integrating Service-Enabled Application Modules chapter of the Developing Fusion Web Applications with Oracle Application Development Framework.

- For more information about testing deployed SOA composite services, see Managing SOA Composite Application Instances in the Administering Oracle SOA Suite and Oracle Business Process Management Suite.

37.6 Troubleshooting the Use Case

Following are tips that may help resolve common issues that arise when developing or running this use case.

To get more logging information, add logger reference oracle.soa.bpel.entity.
### 37.7 What You May Need to Know About Manipulating Back-end Data from a SOA Composite

Before you implement these design patterns, you may want to know more about when variables flush changes back to the business component, how to test variables without invoking ADF Business Components, support for XPath operations, and how to invoke multiple services.

#### 37.7.1 When Entity Variables Flush Changes Back to ADF Business Components

There are three points within the flow when an entity variable flushes its changes back to the ADF Business Components service:

- When the BPEL instance dehydrates. This happens whenever a breakpoint activity is reached (for example receive, onMessage, wait, onAlarm) or execution reaches the end of the flow definition.

- When the BPEL instance dehydrates due to reaching its activity processing threshold. By default, the threshold is set to 600 activities. The property dspMaxRequestDepth in the file bpel-config.xml sets the threshold.

- When execution reaches the end of the scope, if the entity variable has been declared locally (for example within a scope). If the entity variable is declared globally, it flushes changes when the instance completes.

To illustrate, Example 37-3 includes a locally declared entity variable.

Because the variable was defined locally within the scope myScope, after that scope completes, the variable declaration is no longer accessible from the outer enclosing scope. At this point, changes made to the variable from the Assign activity will be flushed to the ADF Business Components service.

Example 37-4 shows an entity variable defined globally, hence its scope will complete when the instance completes.

**Example 37-3  Local Entity Variable**

```xml
<scope name="myScope">
  <variables>
    <variable name="orderEVar" element="fdsm:orderInfo2SDO"
      bpelx:entity.si="OrderService" />
  </variables>
  <bpelx:bindEntity name="BindEntity_1" variable="orderEntityVar">
    <bpelx:key keyname="ns4:OrderId">bpws:getVariableData('inputVariable','payload','/client:BPELEntityVarADFBCProcessRequest/client:orderID')</bpelx:key>
  </bpelx:bindEntity>
  <assign> ... </assign>
</scope>
```

**Example 37-4  Global Entity Variable**

```xml
<process ...
  <variables>
    <variable name="orderEVar" element="fdsm:orderInfo2SDO"
      bpelx:entity.si="OrderService" />
  </variables>
  <sequence>
    <receive ...>
```
37.7.2 Support for XPath Operations

XPath operations can be used in Assign activities to manipulate data. Most XPath operations are supported. Following are noted limitations:

- Nested predicates, such as `-nameStep1[x1[y1>3]]` are not supported.
- Multiple steps within a predicate, such as `-nameStep1[x1/y1>3]` are not supported.
- There are limitations present in the ADF Business Components SDO binding layer, such as numeric computation and function. However, BPEL’s XPath layer will rewrite the expression into a form supported by ADF Business Components. For example:

  - `nameStep1[x1 > 23 + 45 and y1 = concatenate('a', 'b')]`

All entity variables are stored in the BPEL dehydration store. However, BPEL only stores the key data required for the variable and not the variable in its entirety. When loading these variables upon re-hydration, BPEL runs a find request on the variable to reload the variable data. However, the Oracle ADF web application may have changed the variable, leading to potential loss of data integrity.

You can check the variable for data integrity by running `bpelx:entity.doVersionCheck = "true"`. `doVersionCheck` checks for `ObjectVersionId`. If this returns a different value from that in the BPEL store, a fault is raised.

**Example 37-5** illustrates the verification of variable data integrity.

---

**Note:**

The version fields must exist in the data object returned. If the Xpath query fails, an exception is thrown. Make sure the custom version fields are defined in the XSD and that the names match.

---

**Example 37-5  Checking Variable Data Integrity**

// Checks the variable version.
<variable name="Variable_1" element="ns1:emp" bpelx:entity.si="Service1"
bpelx:entity.doVersionCheck="false"/>

// Enables doing a custom version check on custom fields Sal and Deptno.
<variable name="Variable_1" element="ns1:emp" bpelx:entity.si="Service1"
bpelx:entity.customVersionCheck="ns1:Sal ns10:Deptno "/>
37.7.3 Invoking an ADF Business Components Service and Entity Variables in the Same BPEL Process Service Component

The pattern described in this chapter and the pattern described in Orchestrating ADF Business Components Services can be easily combined.
This chapter describes what a SOA composite application needs to do to access logic implemented as PL/SQL in the database.

When to implement: A SOA composite application needs to access logic implemented as PL/SQL in the database.

Design Pattern Summary: The SOA composite application accesses an ADF Business Components service, which in turn accesses the PL/SQL stored procedure.

Involved components:

- Business component that accesses a PL/SQL stored procedure, and is published as a service.
- SOA composite application which includes a BPEL process service component that accesses the ADF Business Components service.

38.1 Introduction to the Recommended Design Pattern

Oracle Fusion applications may contain stored procedures in the database that a SOA composite application needs to access. The stored procedure must be wrapped by an ADF Business Components service; the BPEL process then accesses the ADF Business Components service.

38.2 Other Approaches

Instead of accessing the stored procedure through an ADF Business Components service, you could use a SOA database binding component. However, this is not allowed because the SOA database binding component does not handle data changes or database schema changes gracefully. In addition, a PL/SQL stored procedure definition cannot be considered a service contract, as there is often some needed extrapolation.

Another alternative is to create a Web service directly on top of PL/SQL. This is not allowed because of security issues in the PL/SQL Web service.

38.3 Example

The sample code for this use case can be downloaded from Oracle SOA Suite samples.
38.4 How to Invoke a PL/SQL Stored Procedure from a SOA Composite Application

Instead of directly accessing the stored procedure, you create a business component that accesses the procedure, you then publish the business component as a SOAP service. The SOA composite application component accesses the ADF Business Components service, which in turn invokes the stored procedure.

**To invoke a PL/SQL stored procedure from a SOA composite application:**

1. Create a business component, including an application module.
2. Write a method in your application module that accesses the PL/SQL stored procedure.
3. Generate the service interface for the business component.
4. Invoke it through a SOAP binding, as described in Orchestrating ADF Business Components Services.

**Related Links**

The following documents provide additional information related to subjects discussed in this section:

- For more information about creating a business component with an application module, see the section "Part II: Building Your Business Services" of the Developing Fusion Web Applications with Oracle Application Development Framework.

- For more information about accessing PL/SQL stored procedures, see the section "Invoking Stored Procedures and Functions" in the chapter "Advanced Business Components Techniques" of the Developing Fusion Web Applications with Oracle Application Development Framework.

- For more information about generating a service interface for a business component, see the chapter "Integrating Web Services Into a Fusion Web Application" in Developing Fusion Web Applications with Oracle Application Development Framework.

38.5 Securing the Design Pattern

You secure this design pattern in the same way you secure a pattern with an ADF Business Components service invoked from a SOA composite application. For details, see Securing the Design Pattern.

Identity propagation is enabled using Application User Sessions. When the application module initializes, an Application User Session is created with the user who is currently logged in (assuming no such Application User Session yet exists). The Application User Session is pushed to the database, making it accessible from PL/SQL.

For information about accessing the Application User Session, see Implementing Application User Sessions.

For information about securing the design pattern, see Securing Web Services Use Cases.
38.6 Verifying the Deployment

To properly verify this design pattern, test your business component, then deploy and test the SOA composite application.

**To verify this design pattern:**

1. Test your Oracle ADF application.

2. Deploy the SOA composite application to the standalone WLS where the SOA infrastructure has been installed. Because you created a published event from the SOA composite application to the ADF Business Components service, the ADF Business Components service need not to also be deployed to the SOA infrastructure.

3. Test the deployed SOA composite service using Oracle Enterprise Manager Fusion Middleware Control Console. Every deployed service has its own test page, so you can quickly test that the service functions as you expect. For more information about using the Fusion Middleware Control Console to test deployed SOA composite applications, see the following chapter:

**Related Links**

The following documents provide additional information related to subjects discussed in this section:

- For more information about testing and debugging Oracle ADF applications, see the chapter "Testing and Debugging ADF Components" of the *Developing Fusion Web Applications with Oracle Application Development Framework*.

- For information about testing the ADF Business Components service, see the chapter "Integrating Web Services Into a Fusion Web Application" in the *Developing Fusion Web Applications with Oracle Application Development Framework*.

- For more information about testing and debugging, see the chapter "Automating Testing SOA Composite Applications" in the *Developing SOA Applications with Oracle SOA Suite*. 

Access a PL/SQL Service from a SOA Composite 38-3
This chapter describes what a SOA composite application needs to do to invoke logic implemented by a Java class.

**When to implement:** A SOA composite application needs to invoke logic implemented by a Java class.

**Design Pattern Summary:** The logic is accessed using a web service that the SOA composite component accesses directly. The logic can be placed within an existing ADF Business Components service, or the Java class can be published as a web service.

**Involved components:**

- Business component that contains a service method, and is published as a service; or a Java class published as a web service.
- SOA composite application that includes a BPEL process service component that accesses the ADF Business Components service or the Java class web service.

### 39.1 Introduction to the Recommended Design Pattern

While the most common implementation of business logic will either be inside the application module of a business component or in a SOA composite application, Oracle Fusion applications may contain business logic implemented in a separate Java class. A SOA composite component within the application may need to access this logic. When the logic has some relation to the entity or view objects defined in a business component, it is best to implement the logic as another service method on the component. You can publish the business component as a service, and the SOA component can access the logic through that service.

In some cases, the application does not contain a business component that accesses the same data as the logic in the Java class. In this case, you can publish the class as a web service, allowing the SOA component to directly access the logic through the service.

### 39.2 Other Approaches

Instead of accessing the logic through a SOAP binding, you could use a BPEL component with a Java activity to invoke some custom Java code.

An alternative is to use the `bpelx:exec` command to programmatically call Java methods. This is the same as using a Java activity, only without a UI. It is recommended to use the `bpelx:exec` command to complete light tasks such as the following:

- Manipulating data.
• Calling self-contained Java classes that do not have dependencies on other system resources.

• Calling functions provided by the BPEL service engine, such as adding an auditTrail component, and so on.

• Calling Enterprise JavaBeans.

WARNING:
Do not use the \texttt{bpelx:exec} command to access the application database or a SOAP service.

To access the application database, use a JCA database adapter. To access a SOAP service, use a BPEL Invoke activity.

Another alternative is to call custom Java classes using custom XPath functions.

Related Links
The following documents provide additional information related to subjects discussed in this section:

• For more information about using the \texttt{bpelx:exec} command, see the chapter "Incorporating Java and Java EE Code in a BPEL Process" in the Developing SOA Applications with Oracle SOA Suite.

• For more information about using XPath functions to call a Java class, see the subsection "Creating User-Defined XPath Extension Functions" in the appendix "XPath Extension Functions" in the Developing SOA Applications with Oracle SOA Suite.

39.3 Example
Download the sample code for this use case from the Oracle SOA Suite samples.

39.4 How to Invoke a Java Class from a SOA Composite Application
Instead of directly accessing the Java class, you can add the logic to a business component and then publish the business component as a SOAP service. The SOA composite application component accesses the ADF Business Components service, which in turn invokes the Java code.

To use a business component:

1. Create a business component, including the required logic as a method on an application module,

2. Generate the service interface for the business component and from the SOA component, invoke the service through a SOAP binding, as described in Orchestrating ADF Business Components Services.

Related Links
For more information about creating a business components, see "Part II: Building Your Business Services" in the Developing Fusion Web Applications with Oracle Application Development Framework.
39.5 Securing the Design Pattern

You secure this design pattern in the same way as you secure a pattern that has an ADF Business Components service invoked from a SOA composite application. For details, see Securing the Design Pattern.

39.6 Verifying the Deployment

To properly verify this design pattern, you should test your business component, then deploy and test the SOA composite application.

To verify this design pattern:
1. Test your Oracle ADF application.

2. Deploy the SOA composite application to the standalone WLS where the SOA infrastructure has been installed. Because you created a published event from the SOA composite application to the ADF Business Components service, the ADF Business Components service need not to also be deployed to the SOA infrastructure.

3. Test the deployed SOA composite service using Oracle Enterprise Manager Fusion Middleware Control Console. Every deployed service has its own test page, so you can quickly test that the service functions as you expect.

Related Links
The following documents provide additional information related to subjects discussed in this section:

- For more information about testing and debugging Oracle ADF applications, see the chapter “Testing and Debugging ADF Components” of the Developing Fusion Web Applications with Oracle Application Development Framework.
- For information about testing the ADF Business Components service, see the chapter ”Integrating Web Services Into a Fusion Web Application” in the Developing Fusion Web Applications with Oracle Application Development Framework.
- For more information about using the Fusion Middleware Control Console to test deployed SOA composite applications, see the chapter ”Automating Testing of SOA Composite Applications” in the Developing SOA Applications with Oracle SOA Suite.

39.7 Troubleshooting the Use Case

Following are tips that may help resolve common issues that arise when developing or running this use case.

When writing custom Java code, you may need to use logging messages for debugging. You can then monitor the log files for troubleshooting information.

39.8 What You May Need to Know About Invoking Custom Java Code from a SOA Composite

If you have already created a business component and the method semantics of the needed logic fit with the other service methods already implemented in the
application module, you should add another method for the needed logic to the application module. If the method relies on entity and or view objects on the business component, then adding the method to the business component is the only approach available.

If you have not created a business component or the method semantics do not fit with the other service methods already implemented in the application module, you may want to create a Java web service instead.
This chapter describes what your ADF application needs to do to assign tasks to users or groups.

**When to implement:** When your ADF application needs to assign tasks to users or groups.

**Design Pattern Summary:** An ADF task flow in a web application contains a page where a user action invokes an ADF Business Components object that performs some logic. Because of this user action, a task must be assigned to another user. For example, an employee uses an ADF web application to submit an expense report. The page used to create the expense report is within an ADF task flow. When the expense report is submitted, a task must be raised to the employee's manager so that they can approve or reject the expense report. To make this happen, when the ADF Business Components object is invoked, it invokes a BPEL process service component that uses a human task service component to assign the task to the manager. After the human task service component is invoked, the manager uses the Worklist application to complete the task, in this case the approval or rejection of the expense report. The Worklist application also uses an ADF task flow to present those pages to the manager.

This design pattern uses BPEL so as to enable orchestration after the task is submitted.

**Involved components:**

- ADF web application that includes an ADF Business Components object and an ADF task flow that contains the UI pages where the end user submits data.
- SOA composite with a mediator component that listens for events raised by the ADF application. The mediator invokes a BPEL service component that uses an included human task component.

**Note:**

BPEL is not a requirement for working with human tasks. However, BPEL is used when orchestrating tasks after the end-user submits the human task, for example to approve or reject forms filled out by the end-user.

Instead of using the worklist application, you could use a custom task application or APIs.

### 40.1 Introduction to the Recommended Pattern

Oracle Fusion applications may need to assign users tasks that they need to complete. The application needs to notify users of assigned tasks, then provide a way for them to
complete the tasks. The SOA composite project may include a BPEL process that assigns tasks to users as part of the process flow. The human workflow service can be used to accomplish this. The workflow component also includes an out-of-the-box worklist application displaying all the tasks assigned to a particular user or group. You can use the ADF task flow to create UI pages listing pending tasks. These are displayed upon logging in to the worklist application.

To develop this pattern, an ADF application can invoke an SOA composite application, as described in Initiating a SOA Composite from an Web Application. A human task service component can be included in the composite that assigns tasks to users or roles. This component includes a task editor used to design the metadata for the task, an ADF task flow for creating task forms for the human interaction, and the out-of-the-box Worklist application for users to access tasks and act on them. The Worklist application can use an ADF task flow to manage the pages needed to complete the tasks.

This pattern is recommended for the following reasons:

- You can do any required validation of the data in the ADF Business Components layer as opposed to the process.

- You can pass only minimal data from the ADF web application into the process. Using this approach, you can pass just header level information required to route the task, (for example an expense ID, amount, requester). All other information remains in the database and is accessed in the task form by reference. The task form is used to display relevant details to the user. The data is therefore not carried through the process.

Figure 40-1 shows the recommended pattern.
This chapter explains how to implement the recommended pattern.

**40.2 Other Approaches**

There are no other supported approaches to this use case.

**40.3 Example**

Following is an example that illustrates the design pattern.

An Expense application contains an ADF task flow used to create an expense report and submit it for approval. When the user submits the expense report, a BPEL process service component is invoked that contains a human task service component, which assigns the task of approving expense report to the user’s supervisor. The human task service component notifies the supervisor that an expense report must be approved. When the supervisor logs into the Worklist application, he sees notification of an expense report that requires approval. The Worklist application contains an ADF task flow that allows the supervisor to either approve or reject the expense report.

The sample code for this use case can be downloaded from Oracle SOA Suite samples.

**40.4 How to Manage a Human Task Flow from an ADF Application**

There are three high-level steps needed to invoke a human task flow from an ADF web application:
1. Create and deploy the ADF web application that contains the UI necessary to invoke the SOA composite that contains the human work flow.

2. Create the SOA composite that contains a mediator, BPEL process service component, and human task component that will assign the task to the user.

3. Create the ADF task flow based on the human task component. This will provide the UI necessary for the user to resolve the task.

These procedures are detailed in the remainder of this section.

To invoke a human work flow from an ADF application:

1. Create an ADF application that contains a page that will raise an event, as described in How to Initiate a BPEL Process Service Component from an Web Application.

2. Create an SOA composite application component that includes a mediator service component, along with a BPEL process service component that the mediator service component invokes when the ADF Business Components event is raised. Follow the procedures as documented in How to Initiate a BPEL Process Service Component from an Web Application.

3. In the SOA composite application, create a human task service component that uses the payload of the message received from the mediator service component to provide any parameters needed to create the task in the workflow.

When creating the human task service, note the following:

- The task will require details about the assignee, task parameters, and so on. However, you do not need to capture the entire object used to create the task in the task parameters. You only need to include enough information to make the assignment decision and to then query the object using the service interfaces for the ADF Business Components application module.

  For example, in the expense report sample application, you might create parameters for the expense report number, the "submitted by" value, and optionally the amount. This would be the only information needed to dynamically assign the task to the submitter’s manager.

- You can assign tasks statically or dynamically. Make sure that any user to whom the task is assigned is seeded in the identity management store (for more information, see Securing the Design Pattern).

- You should model any email or other notifications as part of the task metadata. For example, you may want to model an email to the supervisor when the task is assigned to them and an email to the initiator when the task is completed.

4. In the BPEL process, add a human task activity and wire it to the human task service component. In the human task activity, specify which BPEL variables map to the data required for the task.

After you create the human task activity, a switch statement is automatically inserted that contains various branches based on the previously specified outcome of the human task. Based on how the outcome impacts the specific use case, additional business logic should be inserted inside the branches.

For example, in the expense report example, you might include a branch that calls the ADF Business Components object to update the state of the expense report to
Approved. This can be done by exposing the ADF Business Components application module as a service and then invoking it from BPEL as a SOAP service. For more information, see Manipulating Back-End Data from a SOA Composite.

5. Create an ADF task flow based on the human task service component. This will be the flow used in the worklist application that displays the pages the user views to complete the task.

Note the following:

- When creating the task flow, select the .task file created in Step 3. This automatically generates a task data control that can access the task parameters and perform actions on the task such as Approve and Reject.

- On the JSP pages, use the drop handlers on the task data control to insert sections such as Task Header, Comments and Attachments, Approval History, and so on.

- Use the ADF Business Components data control to pull in any transactional data, such as line items. You can use the object id (available in task parameters) in the query API for the ADF Business Components.

- Drag and drop any custom actions (for example, approve or reject) on the form. You can also include system actions (for example, escalate, reassign, delegate) on the form.

- Note that by default, the first page in the task flow is sent in email notifications. The buttons on the page are replaced with ReplyTo links for actions such as Approve and Reject, allowing the user to approve or reject the object in the task from the email. You can also model a different page for email approval.

6. Deploy the SOA composite and the ADF task flow for the human task service to the standalone WLS where the SOA infrastructure has been installed.

Deploy the task from the application menu and the SOA composite from the project menu.

Related Links

The following documents provide additional information related to subjects discussed in this section:

- For detailed procedures on creating a human task service, see the chapter "Creating Human Tasks" in the Developing SOA Applications with Oracle SOA Suite.

- For more information about notifying users of changes to task status, see the chapter "Creating Human Tasks" in the Developing SOA Applications with Oracle SOA Suite.

- For more information about creating an ADF task flow based on the human task service component, see the chapter "Creating Human Tasks" in the Developing SOA Applications with Oracle SOA Suite.

40.5 Other Approaches

You can deploy the human task flow to a remote server without SOA infrastructure.
**Related Links**

For more information about deploying human task flows to remote servers without SOA infrastructure, see the section "How To Deploy a Task Display Form to a non-SOA Oracle WebLogic Server" in the chapter "Designing Task Forms for Human Tasks" in the *Developing SOA Applications with Oracle SOA Suite*.

### 40.6 Securing the Design Pattern

**Note:**

To use the ADF Security Wizard, developers must start up JDeveloper with a role profile that supports JAAS-XS security, such as the Oracle CRM Application Developer Role. To deploy an SOA composite to a WLS container, the application deployer must start up JDeveloper using the default role.

The human task service uses Java platform security (JPS) for accessing user/role profile information. JPS supports multiple providers, such as XML and XS.

The default authentication provider in Oracle WebLogic Server is *WLSAuthenticator*, while the authorization provider is based on the JPS policy store.

The default security configuration uses the Oracle WebLogic Server embedded LDAP as the identity store and *system-jazn-data.xml* as the policy store. This configuration is held in the *workflow-identity-config.xml* file, as shown in Example 40-1.

**Note:**

As other applications plan to use XS database as a repository, XML-based providers must synchronize with the XS database whenever changes occur to user or role related data.

**Related Links**

The following documents provide additional information related to subjects discussed in this section:

- For more information about configuring or updating the human workflow Identity Service, see the section "Configuring the Identity Service" in the chapter "Configuring Human Workflow Service Components and Engines" in the *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

- For more information, see the chapter "Configuring the OPSS Security Store" in the *Oracle Fusion Middleware Applications Security Guide*.

**Example 40-1  Identity Service Configuration**

```xml
<ISConfiguration xmlns="http://www.oracle.com/pcbpel/identityservice/isconfig" >
  <configurations>
    <configuration realmName="jazn.com">
      <provider providerType="JPS" name="JpsProvider" service="Identity">
        <property name="jpsContextName" value="default" />
      </provider>
    </configuration>
  </configurations>
</ISConfiguration>
```
40.7 Verifying the Deployment

To properly verify this design pattern, you should test and deploy your ADF application, deploy the SOA composite application and the ADF task from the Worklist application, and then run the ADF application.

To verify this design pattern:

1. Test your Oracle ADF web application. Be sure to enable the ADF Business Components runtime logging by setting the Java VM parameter `jbo.debugoutput=console`. Doing so logs the event and its payload in the JDeveloper log console.

2. Use Oracle Enterprise Manager Fusion Middleware Control Console to view the SOA composite application and ensure it was properly deployed.

3. Run the ADF application and invoke the method that raises the event.

   Alternatively, you can use the Composite initiate page in Enterprise Manager to send a test message to the BPEL process service component. Note that this bypasses the mediator service component and directly calls the BPEL process service component which in turn invokes the human task service component.

   The web service ending point for each of the services in the composite is

   
   http://HOST_NAME:PORT/fabric/application name/composite_name/service_name

   For example:


4. View the ADF Business Components runtime log to view the event and payload.

5. Use Oracle Enterprise Manager Fusion Middleware Control Console to check if an instance of the SOA composite application has been created. The process should be in running state and waiting at the human task step.

6. Log in to the worklist application to view the task created for the supervisor. The worklist application is located at http://hostname/integration/worklistapp/

   Click Approve or Reject to complete this task.

   If successful, the status of the task will be updated on the home page the Worklist application.

7. In Fusion Middleware Control Console, check that the human task has completed. This ensures that the waiting BPEL process can proceed with the subsequent steps.

Related Links

The following documents provide additional information related to subjects discussed in this section:

- For more information on testing and debugging your Oracle ADF web application, see the chapter "Testing and Debugging ADF Components" of the Developing Fusion Web Applications with Oracle Application Development Framework.
• For more information about using Fusion Middleware Control Console, see the chapter "Deploying SOA Composite Applications" of the Developing SOA Applications with Oracle SOA Suite.

• For more information about the ADF Business Components runtime log, see the chapter "Testing and Debugging ADF Components" of the Developing Fusion Web Applications with Oracle Application Development Framework.

• For more information about Fusion Middleware Control Console, see the chapter "Deploying SOA Composite Applications" of the Developing SOA Applications with Oracle SOA Suite.

40.8 Troubleshoot the Use Case

Following are tips that may help resolve common issues that arise when developing or running this use case.

40.8.1 Worklist Notification Locale Does Not Honor the Regional Applications Session Setting

When logging into the worklist application, both the worklist and human task details honor the regional setting in the Applications Core session. However, the notification locale does not honor the regional setting of the Applications Core session.

Notifications cannot consider the Applications Core sessions of the worklist application.

Consider a task being assigned to user jcooper. The user jcooper may not be logged into the worklist at the time, but notifications must be sent to jcooper when the task is assigned.

To do so, implement oracle.bpel.services.workflow.task.INotificationCallback. You can add print statements to the implementation and verify the values that are returned.

40.8.2 Task Does Not Display in Worklist Application

If you are not able to find the task in the worklist application, try the following:

• Login to the worklist application with weblogic as the user. Navigate to the Administration Tasks tab. This will list all the tasks in the system. Locate the task that was created and validate the state (it should be ASSIGNED) and the assignees.

• Alternatively, log in to Oracle Enterprise Manager Fusion Middleware Control Console at http://HOST_NAME:PORT/em and click the instance ID to display an audit track window. In the audit track view, click the BPEL instance and select the Flow-Debug tab to display the BPEL audit trail. Check the following values in the initiateTaskResponse:

  – task:task/task:systemAttributes/task:state: This should be ASSIGNED

  – task:task/task:systemAttributes/task:assignees: This displays the assignees of the task

• Check the following log files and the shell where the server was started to make sure that there are no errors (for information about logging, see Logging).
• Use the web services page http://HOST_NAME:PORT/integration/services/IdentityService/identity to check user and group properties. Users and groups may be seeded differently from what is assumed and may therefore cause unexpected results.

40.8.3 Task Details Do Not Display in the ADF Task Flow

If you are not able to see the human task details when you click on the task in the worklist application, try the following:

• For design time issues, make sure you start JDeveloper using jdev.exe instead of jdevw.exe in a Windows environment. This brings up a console window in the background that shows any error messages or stack traces.

• If you see any errors during deployment of the ADF task flow, check the following:
  – The ADF task flow for the human task service is deployed on the same instance as the SOA server. If it is deployed to a server without SOA infrastructure, make sure to correctly follow the steps for deploying task flows.
  – Check if the taskflow.properties file exists in your project and has the following settings:

```
human.task.lookup.type=LOCAL
```

  – Check that all shared libraries are installed to the WLS instance where the task flow is deployed, including adf.oracle.domain, adf.oracle.domain.webapp, JSF, JSTL and oracle.soa.workflow or any shared library specified in weblogic-application.xml. You can verify this by logging into Oracle WebLogic Server Console as an administrator, and clicking Deployments.

• If you see the task in the task list but you get an error when clicking on the task title, try the following:
  – Determine if there were any deployment errors.
  – Enable logging for ADF as described in Logging. Set the log level to FINE and deploy the task flow again. After you do this, you should see error messages in the log file. $OC4J_HOME/j2ee/home/log/oc4j/log.xml file
  – If you get any errors on the task details screen when performing any operations, check the following log files:
40.8.4 Logging

You can set logging for the Workflow application and for the ADF task flow.

40.8.4.1 Workflow Logging

To enable debug logging for the workflow service, add a new log_handler and logger for oracle.bpel.services in ORACLE_HOME/j2ee/home/config/j2ee-logging.xml as shown in Example 40-2. After these changes are made, you must restart the server. When this logger is added, the logs will be found in MW_HOME/user_projects/domains/<domain name>/config/config.xml.

Example 40-2 Adding a New Log Handler

```xml
<?xml version="1.0" encoding="iso-8859-1"?>

<logging_configuration>
  <log_handlers>
    ...
    <log_handler name="oracle-bpel-services-handler" class="oracle.core.ojdl.logging.ODLHandlerFactory">
      <property name="path" value="../log/wls/bpel/services"/>
      <property name="maxFileSize" value="10485760"/>
      <property name="maxLogSize" value="104857600"/>
      <property name="encoding" value="UTF-8"/>
      <property name="supplementalAttributes" value="J2EE_APP.name,J2EE_MODULE.name,WEBSERVICE.name,WEBSERVICE_PORT.name"/>
    </log_handler>
    ...
  </log_handlers>
  <loggers>
    ...
    <logger name="oracle.bpel.services" level="FINEST" useParentHandlers="false">
      <handler name="oracle-bpel-services-handler"/>
    </logger>
    ...
  </loggers>
</logging_configuration>
```

40.8.4.2 ADF Task Flow Logging

To enable ADF logging for the task flow, add fragments shown in Example 40-3 to MW_HOME/user_projects/domains/DOMAIN/soainfra/config/fmwconfig/servers/AdminServer/logging.xml and restart the server.

MW_HOME/user_projects/DOMAIN/domains/<domain name>/config/config.xml
Example 40-3 Adding Fragments to the Logging File

```xml
<logger name='oracle.adf' level='FINE' useParentHandlers='false'>
    <handler name='wls-domain'/>
    <handler name='console-handler'/>
</logger>

<logger name='oracle.adfinternal' level='FINE' useParentHandlers='false'>
    <handler name='wls-domain'/>
    <handler name='console-handler'/>
</logger>

<logger name='oracle.jbo' level='FINE' useParentHandlers='false'>
    <handler name='wls-domain'/>
    <handler name='console-handler'/>
</logger>
```

40.9 What You May Need to Know About Managing Tasks from an ADF Application

Tasks are linked to the composite instance that created them. When a new version of the BPEL process service component or workflow service component is deployed, any existing composite and associated task instances are marked stale. You can clean up stale composites using Oracle Enterprise Manager Fusion Middleware Control Console. Cleaning up stale composite instances automatically deletes the associated task instances as well.

Related Links

For more information about managing tasks from an ADF application, see the chapter "Managing SOA Composite Application Instances" in the *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.
This chapter describes what to do when you need to work with data from a remote Oracle ADF Fusion Business Service in the format of an ADF Business Components component, such as rendering the data in a UI table, or creating a view link to it.

**When to implement:** When you need to work with data from a remote Oracle ADF Fusion Business Service in the format of an ADF Business Components component, such as rendering the data in a UI table, or creating a view link to it.

**Design Pattern Summary:** Create service-based entity objects and view objects and use these entity objects and view objects as normal ADF Business Components either in your business logic or for rendering on UI pages to simplify the task.

**Involved components:**
- ADF Business Components service that accesses data from a different pillar.
- Entity object based on the ADF Business Components service, and view object on top of the entity object.

### 41.1 Introduction to the Recommended Design Pattern

When you want to work with data from a remote ADF Business Components service, you can create service-based entity objects and view objects to simplify the task. A service-based entity object is an entity object that encapsulates the details of accessing and modifying a row of data from a remote ADF Business Components service. The service-based entity object then can be used in the same way as a normal database-table-based entity object.

### 41.2 Potential Approaches

If the data that you need to work with is always local to you, that is, available in the same database, then use a table-based entity object instead. The service-based entity object or view object will have additional performance overhead since each has an extra service layer.

Instead of wrapping with a service-based entity object or view object, you always can invoke the ADF Business Components service directly. For more information, see [Synchronously Invoking an ADF Business Components Service from an Application](#). As UI components cannot be bound to Service Data Objects (SDOs) directly and an ADF Business Components service only accesses SDOs, using service-based entity objects and view objects is a simpler approach, especially if you just need to invoke a method as part of your business logic and you do not need to bind the input/output of the method to UI components.
41.3 Example

Currently, no example is available.

41.4 How to Create Service-Based Entity Objects and View Objects

Instead of creating an entity object on top of a database schema, you create it based on a WSDL.

To create service-based entity objects and view objects:

1. Ensure that the WSDL file of the targeted ADF Business Components service is available, either from your file system or from a URL.

2. Create a new entity object using the Create Entity Object wizard.

   **Note:**
   Instead of using a database schema object for the entity’s data source, select Service Interface.

3. Create a view object on top of the entity object that you created in the previous step. This step is the same as creating a normal entity object-based view object.

4. Register the targeted service in your connections.xml file (found in Application Resources > Descriptors > ADF META-INF). This entry is needed during runtime to invoke the service. Note that this also requires that the targeted service be hosted and running, since the registration requires the URL of where the service is deployed. Example 41-1 shows sample code used in the connections.xml file.

5. Get the service interface common.jar file from the service provider, and add the file to your library. This is also required during runtime. The common.jar file is generated when the service provider uses an ADF Business Component Service Interface deployment profile to deploy.

**Example 41-1 Sample connections.xml Code**

```xml
<Reference name="[http://xmlns.oracle.com/apps/adfsvc/deptempService/]DeptEmpService" className="oracle.jbo.client.svc.Service" xmlns="">
  <Factory className="oracle.jbo.client.svc.ServiceFactory"/>
  <RefAddresses>
    <StringRefAddr addrType="serviceInterfaceName">
      <Contents>oracle.apps.adfsvc.deptempService.DeptEmpService</Contents>
    </StringRefAddr>
    <StringRefAddr addrType="serviceEndpointProvider">
      <Contents>ADFBC</Contents>
    </StringRefAddr>
    <StringRefAddr addrType="jndiName">
      <Contents>DeptEmpServiceBean#oracle.apps.adfsvc.deptempService.DeptEmpService</Contents>
    </StringRefAddr>
    <StringRefAddr addrType="serviceSchemaName">
      <Contents>DeptEmpService.xsd</Contents>
    </StringRefAddr>
    <StringRefAddr addrType="serviceSchemaLocation">
      <Contents>oracle/apps/adfsvc/deptempService</Contents>
    </StringRefAddr>
  </RefAddresses>
</Reference>
```

Example 41-2 Oracle Fusion Applications Developer’s Guide
Related Links

The following documents provide additional information related to subjects discussed in this section:

- For more information about creating service-based entity objects and view objects, see the section "Accessing Remote Data Over the Service-Enabled Application Module" in the chapter "Integrating Service-Enabled Application Modules" in Developing Fusion Web Applications with Oracle Application Development Framework.

- For more information, see the chapter "Defining SQL Queries Using View Objects" in Developing Fusion Web Applications with Oracle Application Development Framework.

41.5 Securing the Design Pattern

For more information about securing the use case, see Securing Web Services Use Cases.

41.6 Verifying the Deployment

To properly verify this design pattern, test your service-based view object in an application module tester, then deploy and test the application that uses the service-based entity object and view object.

To verify this design pattern:
Test your Oracle ADF application.

Related Links

For more information about testing and debugging Oracle ADF applications, see the chapter "Testing and Debugging ADF Components" in the Developing Fusion Web Applications with Oracle Application Development Framework.

41.7 Troubleshooting the Use Case

Currently, there are no tips for troubleshooting the use case.

41.8 Understanding the Transactional Behavior of Service-Based Entity Objects and View Objects

Note that the underlying data source for service-based entity objects and view objects is an ADF Business Components service, which is stateless and in a different transaction. When you commit your transaction of your local application module, a service call will be made if there is any change in the service-based entity object that is not part of your local transaction. If the service invocation fails, then the local transaction also will fail. However, if the service invocation succeeds and then later your local transaction fails, it is your error handling code that must perform a
compensating transaction against the remote service to "undo" the previous change made.

41.9 Known Issues and Workarounds

Known limitations are the following:

- A service-based entity object cannot be referenced as a secondary entity usage in a view object.
- A service-based view object cannot have secondary entity usages.

These limitations mean that you cannot create a flattened join of multiple entities if one of them is a service-based entity object. The workaround is to use a view link to traverse from one view object to another.
This chapter describes what a SOA composite application needs to do to invoke an asynchronous service such as an Oracle ADF service or another SOA composite application.

When to implement: A SOA composite application needs to invoke an asynchronous service such as an Oracle ADF service or another SOA composite application.

Design Pattern Summary: A SOA composite is designed with a Mediator or BPEL component that invokes an asynchronous service endpoint, after which goes into a state of rest until the asynchronous endpoint calls back with the response payload.

Involved components:

- One SOA composite (Consumer role).
- One or more (Producer role) SOA Composites or Oracle ADF Services.

42.1 Introduction to the Recommended Design Pattern

Oracle Fusion web applications often include services with long-running computation that are exposed through service interfaces. When invoking services through a synchronous service interface, the service may not execute as desired due to time outs or lack of resources caused by blocked, waiting threads. To solve this problem, services are exposed through asynchronous interfaces that wait for a response when invoking long-running services.

The recommended approach to asynchronous invocation is to create an asynchronous SOA composite with a BPEL process that invokes the asynchronous service.

Note:

Per Oracle Fusion Applications standards, a composite that exposes a synchronous interface must not invoke an asynchronous service.

Figure 42-1 illustrates the process flow of BPEL process invoking an asynchronous service.
The client BPEL process invokes an asynchronous service through the WSDL partner link. The service runs as required, and returns a response to the waiting client BPEL process.

### 42.2 Other Approaches

Following are alternative, unsupported approaches to the use case.

- Invoke services synchronously. This approach is not supported due to time out issues.
- Invoke services asynchronously, without registering a callback handler service. This approach is not supported, as a callback handler is required when asynchronously invoking a service.
- Invoke a one-way service. This approach is not supported for Oracle ADF services.
- Invoke a service asynchronously, registering another service as the callback handler. This approach is not supported. Use business events instead.

**Caution:**

These approaches are not supported and should not be implemented.

### 42.3 Example

An interface table is populated by third-party entities using legacy interfaces, such as FTP and files. After the interface table is populated, a periodic Oracle Enterprise Scheduler job runs, checking for new interface table rows. If the job finds new table rows, it raises a business event that initiates a BPEL process. The BPEL process orchestrates any required services such as those used to import and notify users, obtain necessary approvals, and so on.

In this scenario, one or more services or tasks require several minutes or even hours to complete, as is typical among asynchronous service interfaces. The BPEL process invokes the service and enters a dormant state in which the process progress and variable data are stored in the database, which frees memory and thread resources for other BPEL processes. When the long-running service completes, the asynchronous callback revives the process. The BPEL process continues from where it left off.

You can find the sample code for this use case in the *Oracle SOA Suite samples*. 
42.4 How to Invoke a SOA Composite Application from Within a SOA Composite Application

To initiate an asynchronous service from a BPEL process, do the following:

1. Create a SOA composite with a BPEL process.
2. Create the service reference to the asynchronous web service you want to invoke.
3. Populate the BPEL process with scope, invoke, receive, assign and fault-handling activities.

Before defining the service reference, create your composite and BPEL process with the requisite input and output payload types. Figure 42-2 shows an example of a minimalist composite with a new BPEL process.

Figure 42-2 Composite Before Web Service Reference Definition

Defining the service reference to the asynchronous web service endpoint involves the following tasks:

1. Define the new web service reference.
2. Wire the BPEL process to the new service reference.
3. Invoke the asynchronous web service from the BPEL flow.
4. Deploying and testing the composite.

42.4.1 Define a New Web Service Reference

Define the service that the composite is to invoke.

To define the new web service reference, do the following:

1. In the composite, define the asynchronous service reference by dragging a Web Service component to the right-hand swim lane of the composite.

   The Create Web Service dialog opens.

2. In the Create Web Service dialog, enter the following information:
   - Name: Enter the name of the web service reference for this composite.
• **Type:** Select Reference. A Reference is a service that this composite will be invoking.

• **WSDL URL:** Select the full URL to the WSDL of the asynchronous web service to be invoked. Use the service explorer to navigate to and select the WSDL.

• **Port Type:** Select the execute port type of the end point service. This value is often automatically defaulted by the UI. This is the port type invoked by the composite.

• **Callback Port Type:** Select the callback port type of the end point service. This value is often defaulted automatically by the UI. This is the port type used to call back the composite.

Figure 42-3 shows an example of a filled in Create Web Service dialog.

**Figure 42-3 Create Web Service Dialog**

![Create Web Service Dialog](image)

### 42.4.2 Wiring the BPEL Process to the New Web Service Reference

To invoke a web service, a BPEL process must include a local partner link definition. You can define a partner link in one of two ways.

- Define a partner link in the BPEL process editor.
- Drag the interface pin from the BPEL process to the service reference.

Defining a partner link involves additional work as compared to the alternative approach of dragging the interface pins. However, when creating a partner link for a web service that has not yet been defined as a service reference in the composite, Oracle JDeveloper automatically creates the composite service reference and wires the BPEL process to that service.

Dragging the interface pin from a component to a service is a quick and easy way to automatically create partner links and other service component integrations, such as wiring a Mediator to a Business Rule component. Wiring components by clicking and dragging automatically generates the metadata files and entries used to support the interaction between the components.
To define a partner link in the BPEL process editor:

1. In the Composite Editor, double-click the BPEL process component.

2. In the right-hand swim lane of the BPEL process, right-click the area under Partner Links.
   The Create Partner Link dialog is displayed, as shown in Figure 42-4.

   **Figure 42-4  Create Partner Link Dialog**

   ![Create Partner Link Dialog](image)

3. In the Create Partner Link dialog, enter the following information:
   - **Name**: Enter the name of the partner link. This defaults to the service name.
   - **WSDL URL**: Enter the full URL for the WSDL file of the asynchronous web service to be invoked.
   - **Partner Link Type**: Enter the service partner link type. This value is automatically filled in using the service name, as derived from the WSDL referenced in the WSDL URL field.
   - **Partner Role**: From the dropdown list, select the `ServiceNameProvider` role. This is the role of the producer service for which you entered the WSDL.
   - **My Role**: From the dropdown list, select the `ServiceNameRequester` role. This is the role of the BPEL process as the consumer of the asynchronous service.

4. Alternatively, click and drag the pin from the BPEL component to the service reference.
   a. Hover the mouse over the BPEL component and the input and output icons.
   b. Drag the input and output icons to the matching icons on the service reference component, as shown in Figure 42-5.

   ![Drag and Drop Partner Link](image)
42.4.3 Invoke the Asynchronous Web Service from the BPEL Flow

By default, an asynchronous BPEL flow contains two activities: the receive activity that starts the process and the invoke activity that initiates the callback response.

Asynchronous services do not throw faults, such that they must always return a valid payload whenever possible. In the event of a business failure at the endpoint service, services should return a payload that contains a failed status.

Invoking the asynchronous web service from the BPEL flow involves the following steps:

1. Add required resilience and logging.

2. Add a scope activity to include the asynchronous service invocation and variable assignment activities.

3. Add fault handlers to trap any system failures that may occur during any activities within a scope, so as to permit logging, incident creation and error management as required by the use case. Asynchronous services do not throw WSDL-defined faults, necessitating a fault handler.

4. Add logging. All BPEL processes within Oracle Fusion web applications are required to implement translatable, tokenized, message-level logging for fault-handling scenarios.

5. Invoke the service.

6. Add invoke, receive and assign activities.

   a. Invoke: This activity defines the partner link and invoke operation, and defines the BPEL variable containing the payload to be submitted.

   b. Receive: This activity defines the partner link and operation to be invoked, and defines the variable to store the response payload when the asynchronous callback is made.

   c. Assign (1): The first assign activity copies BPEL process input data, XML fragments and XPath expression results to the payload that is sent to the asynchronous service.

   d. Assign (2): The second assign activity copies the contents of the asynchronous response payload to the BPEL output variable, which is then sent back to the calling process.
To invoke the asynchronous web service from the BPEL flow:

1. To adhere to Oracle Fusion Applications fault handling and logging standards, add the following activities:
   - **Scope:** By default, the main BPEL flow is the top-level scope of the process. However, should any of the activities generate a fault within that scope, compensation or recovery options are limited. Leverage nested scopes to contain errors and implement resilient functionality.
   - **Fault Handlers:** Each nested and top-level scope should have qualified and catch-all fault handlers to catch business- or system-level faults and respond accordingly to the use case.
   - **Logging:** Add assign activities as needed. Alternatively, add copy operations to pre-existing assign activities to implement logging.

2. Add a scope activity.
   a. In the Composite Editor, double-click the BPEL component to open the BPEL process editor.
   b. Drag a new scope activity onto the process and name according to the activities it will contain, for example, `InvokeAsyncServiceName`.
   c. To the scope activity, add the service invocation and variable assignment activities.

3. Add a catch-all fault handler to trap any errors that may occur during assign, invoke, receive or dehydration activities that may occur within the scope. Use either of the following:
   - **Reply Normally:** Returns a payload with an error status and completes in all cases.
   - **Wait State:** Sends the asynchronous response, which then sends a user notification or updates an Oracle ADF record, and enters a correlated wait state. The wait state allows another entity to invoke the in-flight process that provides additional instructions.

4. Implement translatable, tokenized, message-level logging for fault-handling scenarios. Optionally, implement bread crumb logging and incident creation, where applicable to the use case.

5. Define an invoke activity for the asynchronous web service. In the scope previously defined, drag an **Invoke** activity to the flow.

6. Double-click the activity. In the Invoke Activity dialog, enter the following information.
   - **Name:** The name of the invoke activity, such as `Invoke<ServiceName>`.
   - **Interaction Type:** Select **Partner Link**.
   - **Partner Link:** Select a partner link from the list of partner links that were previously defined in the BPEL process metadata. To select a partner link, browse through the expanded list of partner links for the asynchronous service defined earlier.
• **Operation:** Select an operation from the list of all the available synchronous and asynchronous operations defined on the web service of the partner link. Be sure to select the relevant asynchronous operation.

**Note:**
For Oracle ADF Services, the operation names end with `Async`.

• **Input:** Click the **Add** button to create a scope-local variable to contain the payload intended for the asynchronous service invocation. **Figure 42-6** shows a completed Invoke Activity dialog.

**Figure 42-6  Invoke Activity Dialog**

![Invoke Activity Dialog](image)

7. Define a receive activity to handle the asynchronous callback from the endpoint service, and store the response payload in a variable for use by the process. Drag a receive activity immediately following the invoke activity.

8. In the Receive Activity dialog, enter the following information.
   - **Name:** Enter a name for the invoke activity, such as `Invoke<ServiceName>`.
   - **Interaction Type:** Select **Partner Link**.
   - **Partner Link:** Select a partner link from the list of partner links that were previously defined in the BPEL process metadata. To select a partner link, click the **Browse** button and browse through the expanded list of partner links for the asynchronous service defined earlier.
   - **Operation:** Select an operation from the list of all the available synchronous and asynchronous operations defined on the partner link’s web service. Be sure to select the appropriate asynchronous callback operation.
• **Variable:** Click the **Add** button to define a scope-local variable to contain the payload received from the asynchronous service invocation.

**Figure 42-7** shows a completed Receive Activity dialog.

**Figure 42-7 Receive Activity Dialog**

9. Drag an assign activity onto the flow (within the scope) just above the invoke activity you created earlier. Drag a second assign activity onto the flow just below the receive activity you created earlier.

10. Name each assign activity.

   Double-click the activity, click the **General** tab and enter a meaningful name such as `AssignInputForAsync<ServiceName>` or `CopyOutputFrom<ServiceName>`.

11. Select the **Copy Operation** tab and click the **Add** button to open the Create Copy Operation dialog.

   In the Create Copy Operation dialog, select **Copy Operation**.

12. Depending on the type of data you want to assign to the invoke input variable, configure the following.

   • **Variable:** Select the contents of an element from the source variable, such as the default BPEL `inputVariable`, as shown in **Figure 42-8**.
**Figure 42-8  Variable Copy Operation**

- **XPath Expression**: Enter an XPath expression as required, such as `concat($variable1, '-', $variable2)`, as shown in Figure 42-9.

**Figure 42-9  XPath Expression Copy Operation**
• **XML Fragment**: A fully namespace-qualified XML document that matches the XML schema structure of the target variable or variable element, as shown in Figure 42-10.

**Note:**
If you receive an "Invalid XML" error message, ensure that all elements are namespace qualified within the context of the fragment as BPEL process namespaces are not scoped into fragments.

*Figure 42-10  XML Fragment Copy Operation*

• **Partner Link**: The contents and properties of a Partner Link (useful for dynamic partner links).

13. Save your files.
14. Deploy the SOA composite application to the SOA infrastructure.
15. Test the application.

**Related Links**
The following documents provide additional information related to subjects discussed in this section:

- For more information about deploying SOA composite applications, see *Deploying SOA Composite Applications* in *Developing SOA Applications with Oracle SOA Suite*.
- For more information about testing your SOA composite application, see the chapter *Automating Testing of SOA Composite Applications* in *Developing SOA Applications with Oracle SOA Suite*.
42.4.4 What Happens When You Invoke an Asynchronous Service from within a SOA Composite Application

Defining the web service reference generates a local, abstract WSDL (.wsdl) file named after the service for which it was created. This WSDL file contains partner link information used by BPEL, as well as a reference to the asynchronous web service WSDL URL for looking up message type and schema information. Typically, the WSDL file name is the same as the service name, with the suffix .wsdl appended at the end.

42.4.5 What Happens at Runtime: How an Asynchronous Service is Invoked from within a SOA Composite Application

At runtime, the following occurs within the BPEL process:

- The invoke activity initiates the remote service.
- The BPEL process is dehydrated (session and variable data is serialized) to the database.
- The process thread is freed such that other potential processes may use it while waiting for the asynchronous service to complete and respond.

42.5 Securing the Design Pattern

To secure this pattern, add the Oracle Web Services Manager policies to both the service endpoint and service reference components in the composite.

For more information, see Securing Web Services Use Cases.

42.6 Verifying the Deployment

To properly test this pattern, deploy this SOA composite to the SOA domain and initiate the composite through the service test client page.

To verify the deployment:

1. Open a web browser and navigate to the following URL format.
   
   http://host:port/em

   Note:

   Use the HTTPS protocol instead of HTTP if your server is configured to use SSL.

   You will see a list of all successfully deployed composites in this SOA environment.

2. Click the service endpoint link beneath the name of the deployed composite. This renders the service test client page, which then can be used to enter payload data and invoke the composite.

3. Enter the payload fields as needed and click Initiate.
4. Navigate to Oracle Enterprise Manager Fusion Middleware Control Console using the following URL format.
   
   http://host:port/em

5. Login to Oracle Enterprise Manager Fusion Middleware Control Console and take the following actions.
   
   a. On the left side of Oracle Enterprise Manager Fusion Middleware Control Console, expand the SOA tree node.
   
   b. Expand the **soa-infra** node for the correct SOA server and select the name of the deployed composite.
   
   c. Click the instance ID link for the most recent instance of this composite. The Flow Trace window displays.
   
   d. In the Flow Trace window, find the entry for your BPEL process component. Click the BPEL process component to open the Audit Flow view.

   The Audit Flow view displays the activities that were executed in the BPEL process, along with payload details that you can view by clicking the activity.

42.7 Troubleshoot the Use Case

Following are tips that may help resolve common issues that arise when deploying or running this use case.

42.7.1 Deployment

If failures occur during compilation or deployment, observe the Oracle JDeveloper console and compiler output to resolve any issues.

If deployment is successful but the composite does not display in the soa-infra composites list, check the server's diagnostic log and console output for any exceptions and resolve them.

42.7.2 Runtime

If faults occur when invoking the composite, the logging activities and fault-handling branches should provide meaningful content in the applications diagnostic log (defined in **logging.xml**) or be present in the callback payload.

Use the Audit Flow view to diagnose the problem and correct your BPEL process, then redeploy. For more information about using the Audit Flow view, see the deployment steps in Verifying the Deployment.

42.8 What You May Need to Know About Invoking an Asynchronous Service from Another SOA Composite

Make sure to finalize the XML schemas before implementing the design pattern and defining payload types and variables.
Synchronously Invoking an ADF Business Components Service from an Oracle ADF Application

This chapter describes what to do when you need to invoke an ADF Business Components service either from an ADF Business Components object or from a UI. Use only with synchronous processes with an immediate response.

**When to implement:** When you need to invoke an ADF Business Components service either from an ADF Business Components object or from a UI. Use only with synchronous processes with an immediate response.

**Design Pattern Summary:** Use `ServiceFactory` to generate a dynamic proxy to the target ADF Business Components service, then use the proxy to invoke the desired service method.

**Involved components:**
- ADF Business Components service
- A local ADF Business Components object (such as an application module or an entity object), or from a UI via either a local ADF Business Components object or a managed JavaBean

### 43.1 Introduction to the Recommended Design Pattern

When you need to invoke an ADF Business Components service from an Oracle ADF application, use `oracle.jbo.client.svc.ServiceFactory` to invoke the service.

### 43.2 Potential Approaches

One alternative is to use JAX-WS to generate static proxy classes. This is prohibited for ADF Business Components services because:

- Using JAX-WS generates many proxy classes that you need to maintain, which increases maintenance costs.
- `oracle.jbo.client.svc.ServiceFactory` uses a local Java call if the service is co-located, which offers performance benefits.

The other alternative is to use a dynamic invocation interface. This is not recommended as it requires much more coding.

If you need to bind the input or output of the service to UI components, consider using service-based entity objects and view objects. For more information, see *Working with Data from a Remote ADF Business Components Service.*
43.3 Example

Currently, no example is available.

43.4 How to Invoke an ADF Business Components Service from an Oracle ADF Application

On the service provider side, you must create an ADF Business Components service interface deployment profile. This profile generates two JAR files: one common JAR file that contains only the service interface, and another file that contains the implementation.

To invoke an ADF Business Components service from an Oracle ADF application:

1. Navigate to Application Resources > Descriptors > ADF META-INF and open the connections.xml file.

   This entry is needed to invoke the service during runtime.

   **Note:**

   The targeted service must be hosted and running, as registration requires the URL of the service deployment location.

2. Register the targeted service in connections.xml, as shown in Example 43-1.

3. Get the service interface common JAR file from the service provider and add it to your library.

   This file is required during runtime. The common JAR file is generated when the service provider uses a ADF Business Components service interface deployment profile for deployment.

4. In your ADF Business Components service or managed Java bean class, invoke the desired service using ServiceFactory, as shown in Example 43-2.

   Sample queries are shown in Example 43-3.

   **Example 43-1  Sample Code for the File connections.xml**

   ```xml
   <Reference name="{http://xmlns.oracle.com/apps/adfsvc/deptempService/}DeptEmpService" className="oracle.jbo.client.svc.Service" xmlns="">
   <Factory className="oracle.jbo.client.svc.ServiceFactory"/>
   <RefAddresses>
     <StringRefAddr addrType="serviceInterfaceName">
       <Contents>oracle.apps.adfsvc.deptempService.DeptEmpService</Contents>
     </StringRefAddr>
     <StringRefAddr addrType="serviceEndpointProvider">
       <Contents>ADFBC</Contents>
     </StringRefAddr>
     <StringRefAddr addrType="jndiName">
       <Contents>DeptEmpServiceBean#oracle.apps.adfsvc.deptempService.DeptEmpService</Contents>
     </StringRefAddr>
   </RefAddresses>
   </Reference>
   ```

43-2 Oracle Fusion Applications Developer's Guide
Example 43-2  ServiceFactory Code

```java
OrganizationService svc = (OrganizationService)ServiceFactory.getServiceProxy(OrganizationService.NAME);
List orgs = new ArrayList(2);
Org org1 = (Org)DataFactory.INSTANCE.create(Org.class);
org1.setOrganizationId(new Long(10000));
org1.setOrgName("OrgName");
org1.setName("TranslatedName"); //... and set more attributes
orgs.add(org1);
svc.processOrganization("Merge", orgs, null);
```

Example 43-3  Query Samples

```java
// Retrieve only Dname, Loc from Dept and exclude Empno from Emp.
DeptEmpService mSvc = (DeptEmpService)ServiceFactory.getServiceProxy(DeptEmpService.NAME);
FindCriteria fc = (FindCriteria)DataFactory.INSTANCE.create(FindCriteria.class);
List l = new ArrayList();
l.add("Dname");
l.add("Loc");
l.add("Emp");
fc.setFindAttribute(l);
List cfcl = new ArrayList();
ChildFindCriteria cfc =
(ChildFindCriteria)DataFactory.INSTANCE.create(ChildFindCriteria.class);
cfc.setChildAttrName("Emp");
List cl = new ArrayList();
cl.add("Empno");
cfc.setFindAttribute(cl);
cfc.setExcludeAttribute(true);
cfcl.add(cfc);
fcc.setChildFindCriteria(cfcl);
DeptResult res = mSvc.findDept(fc, null);

// Exclude PurchaseOrderLine from PurchaseOrder.
PurchaseOrderService svc =
(PurchaseOrderService)ServiceFactory.getServiceProxy(PurchaseOrderService.NAME);
FindCriteria fc = (FindCriteria)DataFactory.INSTANCE.create(FindCriteria.class);
List l = new ArrayList();
fcc.setExcludeAttribute(true);
l.add("PurchaseOrderLine");
fcc.setFindAttribute(l);
PurchaseOrderResult res = svc.findPurchaseOrder(fc, null);

// Retrieve only the 2nd Emp along with Dept with Deptno=10.
FindCriteria fc = (FindCriteria)DataFactory.INSTANCE.create(FindCriteria.class);
```
// Create the view criteria item.
List value = new ArrayList();
value.add(new Integer(10));
ViewCriteriaItem vci =
    (ViewCriteriaItem)DataFactory.INSTANCE.create(ViewCriteriaItem.class);
vci.setValue(value);
vci.setAttribute("Deptno");
List<ViewCriteriaItem> items = new ArrayList(1);
items.add(vci);
// Create view criteria row.
ViewCriteriaRow vcr = (ViewCriteriaRow) DataFactory.INSTANCE.create(ViewCriteriaRow.class);
vcr.setItem(items);
// Create the view criteria.
List group = new ArrayList();
group.add(vcr);
ViewCriteria vc = (ViewCriteria)DataFactory.INSTANCE.create(ViewCriteria.class);
vc.setGroup(group);
// Set filter.
fILTER(vc);

List cfcl = new ArrayList();
ChildFindCriteria cfc =
    (ChildFindCriteria)DataFactory.INSTANCE.create(ChildFindCriteria.class);
cfc.setChildAttrName("Emp");
cfc.setFetchStart(1);
cfc.setFetchSize(1);
cfcl.add(cfc);
fc.setChildFindCriteria(cfcl);
DeptResult dres = svc.findDept(fc, null);
pw.println("### Dept 10 and 2nd Emp ###");

Related Links
The following documents provide additional information related to subjects discussed in this section:

- For more information about invoking ADF business components services from an Oracle ADF application, see the chapter "Integrating Service-Enabled Application Modules" in Developing Fusion Web Applications with Oracle Application Development Framework.

- For more information about integrating service-enabled application modules, see the chapter "Integrating Service-Enabled Application Modules" in Developing Fusion Web Applications with Oracle Application Development Framework.

43.5 Securing the Design Pattern
For more information about securing the use case, see Securing Web Services Use Cases.

43.6 Verifying the Deployment
To properly verify this design pattern, test your ADF Business Components object in an application module tester or a UI.

To verify this design pattern:
Test your Oracle ADF application.
Related Links

For more information about testing Oracle ADF applications, see the chapter “Testing and Debugging ADF Components” in the Developing Fusion Web Applications with Oracle Application Development Framework.
Implement an Asynchronous Service Initiation with Dynamic UI Update

This chapter describes what to do when initiating asynchronous or long-running functionality from an Oracle ADF UI and, on completion, notifying users of the completion of that process by dynamically updating the UI. This provides a more dynamic experience for the end user and eliminates the need to constantly click a refresh button.

Note:

- To date, the only supported Oracle ADF UI components for Active Data Service (ADS) update are outputText and image.
- This pattern is in the process of being re-written for conformance to Oracle WebLogic Server and Oracle Java Messaging Service (JMS) guidance and, in the interim, should be implemented with the understanding that a re-write is pending.

When to implement: When initiating asynchronous or long-running functionality from an Oracle ADF UI and, on completion, notifying users of the completion of that process by dynamically updating the UI. This provides a more dynamic experience for the end user and eliminates the need to constantly click a refresh button.

Design Pattern Summary: Oracle ADF UI registers an Active Data control subscriber on top of a JMS queue. The Oracle ADF UI then raises a business event either via the default CRUD operations on the entity or programmatically via Java. This event initiates a BPEL process that performs work and, when completed, invokes a synchronous ADF Business Components service method to trigger pushing the message on the JMS queue, which then causes the Active Data Service control to refresh the component or area of the component.

Involved components:

- Oracle ADF UI
- ADS
- JMS
- Business Events (programmatic)
- SOA Mediator
- SOA BPEL
• ADF Business Components services

44.1 Introduction to the Recommended Design Pattern

Asynchronous services cannot be invoked from Java code in Oracle Fusion applications. When notification of completion of asynchronous, long-running functionality is required in a UI, business events can be used for asynchrony. In addition, ADS triggered over JMS will cause the UI update when the BPEL process completes and invokes the ADF Business Components service to signal its completion.

This approach is recommended because supported technology is used. The approach also supports dynamic page updates if the user navigates away and later returns.

44.2 Potential Approaches

Other than the Oracle ADF UI > Event > BPEL > ADF Business Components > ADS approach, following are the potential approaches:

- Invoke asynchronous service and wait for an asynchronous callback. Unsupported due to thread blocking, pooled resources checked out until functionality completes and returns, potentially hours and days.
- Invoke asynchronous functionality through synchronous interface. Potential time out issues and thread blocking caveats.
- JAX-WS proxy to synchronously invoke the one-way initiate operation, register an ADF Business Components service as the callback service. This is a bit of a hack, not extendible as an integration point, and it doesn't dynamically update the UI.

44.3 Example

The following is an example that illustrates the design pattern.

In an order workbench, an end user selects an order and submits it to a scheduling system for fulfillment. The scheduling system services take several seconds to several minutes to acknowledge scheduling and when the user clicks the button to initiate the scheduling process, must be notified in the UI upon successful scheduling for fulfillment without the need to repeatedly refresh the page by hand.

In this implementation, entering the UI data and clicking Schedule programmatically raises a business event, initiates a BPEL process which goes through processing and approvals as needed, then finally invokes an order ADF Business Components service to complete the process and publish the JMS message to trigger the ADS UI update.

44.4 How to Implement an Asynchronous Service Initiation with Dynamic UI Update

To enable the UI for dynamic update via ADS, you must first create the ADS handler, which uses a common set of JMS Queue handlers to broker the updates coming from the call to ADF Business Components services.

The main steps are as follows, as shown in Figure 44-1:

1. A business event is raised to initiate BPEL, which can perform work asynchronously.
2. The BPEL process updates the database and submits Oracle Enterprise Scheduler jobs.
3. At the end of the BPEL process, a web service is invoked to publish the message.
4. The web service publishes the message to JMS.
5. JMS delivers the message to the ActiveDataCollectionModel.
6. The ActiveDataCollectionModel decodes the message and updates the UI.

Figure 44-1 Technology Flow Diagram (with Optional Oracle Enterprise Scheduler Job Submission)

Prerequisites (for Prototyping Only)
Create the JMS queue in your development environment. Use the prototype common library to build this functionality into your application with minimal changes after the dependent functionality is consumed by the infrastructure.

For prototyping only, take the following steps to set up JMS using Oracle WebLogic Server Console:

---

**Note:**
This procedure assumes that the JMS Module does not already exist.

1. In the Oracle WebLogic Server Console, navigate to **Messaging > JMS Modules**.
2. Click **New** to create the JMS Module.
3. Name the module "FusionAppsADSJMSModule" and click **Next**.
4. In the Targets panel, choose the **AdminServer** target and click **Next**.
5. Choose "Would you like to add resources to this JMS system module?" and click **Finish**.
6. In the Summary of Resources table, click **New**.
7. Choose **Connection Factory** and click **Next**.
8. Name the connection factory **FusionAppsADSJMSConnectionFactory**, provide the JNDI name **jms/Prototype/MyQueueConnFactory**, and click **Next**.
9. Click **Finish**.
10. Verify the new connection factory in the Summary of Resources table and click New.

11. Choose Queue and click Next.

12. Name the queue FusionAppsADSJMSQueue, provide the JNDI name jms/Prototype/MyQueue and click Finish.

44.4.1 Writing the Active Data Handler

The classes shown in Example 44-1, Example 44-2, and Example 44-3 are common to every implementation of this pattern, and are responsible for handling JMS integration with ADS supported events.

**To implement the Active Data Collection Model:**

The Active Data Collection Model, driven by the ADS infrastructure, manages the messages coming from the queue and propagates them to the UI as Oracle ADF Rich Events. Implement the Active Data Collection Model by extending the CollectionModel class in the org.apache.myfaces.trinidad.model package and overriding the `startActiveData, stopActiveData` and `onMessage` methods. The class must implement `ActiveDataModel` and `MessageListener` as the `onMessage` method accepts JMS messages (which is a list of update events) and runs them through the active data listener.

---

**Note:**

Instead of implementing all the logic for CollectionModel, delegate to the collection model returned by the tree binding.

---

What you need to know before you begin:

- The following methods must be implemented for `ActiveDataModel`:
  - `getActiveDataPolicy()` always returns `ActiveDataPolicy.ACTIVE`;
  - `startActiveData(Collection<Object> rowKeys, int startChangeCount, ActiveDataListener listener)` is where you create a queue receiver of the topic subscriber in JMS. If you are not using JMS, this is where you register yourself with the event source as listener.
  - `stopActiveDate(Collection<Object> rowKeys, ActiveDataListener listener)` removes the queue receiver of the topic subscriber in JMS.
  - `getCurrentChangeCount()`: ADS expects the events to arrive in order. Keep a counter in the JavaBean, so that the counter increments when a new event is pushed.

- For `ActiveDataCollectionModel` to be the queue receiver or topic subscriber, `ActiveDataCollectionModel` must implement the `MessageListener` interface using the `onMessage` method. Do the following:
  1. Get the payload from the message. It should be `DataUpdateEvent`. 

---

44-4 Oracle Fusion Applications Developer’s Guide
2. Convert `DataUpdateEvent` to `ActiveDataEvent` so that ADS can process the event.

3. Deliver `ActiveDataEvent` to ADS.

Example 44-4 shows a collection model returned by a tree binding.

There are two reasons for implementing the `getModel()` method this way:

- It is necessary to delegate all collection model-related logic to the model returned by the tree binding. Inside the collection handler, you must get a handle to the collection model returned by the tree binding by looking up the binding container. As you reference the collection model often, store it somewhere for optimal performance. Make sure the managed JavaBean has a view scope while the binding container, tree binding or collection model has a request scope. You cannot store the collection model on the JavaBean. Instead, store the collection model in the request scope. When accessing the collection model, look it up first in the request scope. If the value is null—for example, at the beginning of the request—retrieve the value from the binding container.

- When pushing the `ActiveDataEvent` through ADS, only the UI is updated with the new value. The binding layer is not aware that the underlying data source has changed. If the page is refreshed now, the UI displays the old data from the binding layer. A workaround is to keep a `refreshBinding` flag on the `ActiveDataCollectionModel` to indicate whether the binding requires refreshing. The flag is initially set to false. When an event is received, the flag is set to true. When getting the collection model, check for this flag first. If the flag is set to true, programmatically refresh the related binding before returning the collection model. Example 44-5 shows sample `ActiveDataCollectionHandler` code.

**Example 44-1  DemoDataChangeEntry.java**

```java
package ads.demo.common;

import java.io.Serializable;

public class DemoDataChangeEntry implements Serializable {
    public enum ChangeType {
        /**
         * Indicates the change is row value updates
         */
        UPDATE,

        /**
         * Indicates the change is a new row insertion
         */
        INSERT,

        /**
         * Indicates the change is a new row insertion before a row
         */
        INSERT_BEFORE,

        /**
         * Indicates the change is a new row insertion after a row
         */
        INSERT_AFTER,
    }
}
```

Implement an Asynchronous Service Initiation with Dynamic UI Update 44-5
/**
 * Indicates the change is a new row insertion inside a parent
 */
INSERT_INSIDE,

/**
 * Indicates the change is row deletion
 */
REMOVE,

/**
 * Indicates the change is range refresh
 */
REFRESH

public DemoDataChangeEntry() {
    super();
}

public DemoDataChangeEntry(Object[] pk, ChangeType type,
        String[] attributes, Object[] values) {
    _pk = pk;
    _type = type;
    _attributes = attributes;
    _values = values;
}

private Object[] _pk;
private ChangeType _type;
private String[] _attributes;
private Object[] _values;

public Object[] getPk() {
    return _pk;
}

public ChangeType getType() {
    return _type;
}

public String[] getAttributes() {
    return _attributes;
}

public Object[] getValues() {
    return _values;
}

public void setPk(Object[] _pk) {
    this._pk = _pk;
}

public void setType(ChangeType _type) {
    this._type = _type;
}

public void setAttributes(String[] _attributes) {
    this._attributes = _attributes;
}

public void setValues(Object[] _values) {
Example 44-2  DemoDataUpdateEvent.java

```java
package ads.demo.common;
import java.io.Serializable;
import java.util.List;
public class DemoDataUpdateEvent implements Serializable {
    public DemoDataUpdateEvent() {
    }
    public DemoDataUpdateEvent(List<DemoDataChangeEntry> entries) {
        _entries = entries;
    }
    private List<DemoDataChangeEntry> _entries;
    public List<DemoDataChangeEntry> getEntries() {
        return _entries;
    }
    public void setEntries(List<DemoDataChangeEntry> _entries) {
        this._entries = _entries;
    }
}
```

Example 44-3  JMSHelper.java

```java
package ads.demo.common;
import java.util.Hashtable;
import javax.jms.JMSException;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;
import javax.jms.Queue;
import javax.jms.QueueConnection;
import javax.jms.QueueConnectionFactory;
import javax.jms.QueueReceiver;
import javax.jms.QueueSender;
import javax.jms.QueueSession;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
public final class JMSHelper {
    private static JMSHelper _instance = new JMSHelper();
    public static JMSHelper getInstance() {
        return _instance;
    }
    public ObjectMessage createObjectMessage() throws JMSException {
        return qsession.createObjectMessage();
    }
}
```

Implement an Asynchronous Service Initiation with Dynamic UI Update 44-7
public void sendMessage(ObjectMessage message) throws JMSException {
    qsender.send(message);
}

public QueueReceiver createQueueReceiver(MessageListener listener,
    String messageFilter) throws JMSException {
    QueueReceiver qreceiver = qsession.createReceiver(queue, messageFilter);
    qreceiver.setMessageListener(listener);
    return qreceiver;
}

private JMSHelper() {
    try {
        init();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

// Defines the JMS context factory.
private final static String JMS_QUEUE_FACTORY =
    "jms/Prototype/MyQueueConnFactory";

// Defines the queue.
private final static String ADS_QUEUE = "jms/Prototype/MyQueue";

private QueueConnectionFactory qconFactory;
private QueueConnection qcon;
private QueueSession qsession;
private QueueSender qsender;
private Queue queue;

/**
 * Creates all the necessary objects for sending
 * messages to a JMS queue.
 * @param ctx JNDI initial context
 * @param queueName name of queue
 * @exception NamingException if operation cannot be performed
 * @exception JMSException if JMS fails to initialize due to internal error
 */
private void init() throws NamingException, JMSException {
    InitialContext ctx = new InitialContext();
    qconFactory = (QueueConnectionFactory) ctx.lookup(JMS_QUEUE_FACTORY);
    qcon = qconFactory.createQueueConnection();
    qsession = qcon.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
    queue = (Queue) ctx.lookup(ADS_QUEUE);
    qsender = qsession.createSender(queue);
    qcon.start();
}

/**
 * Closes JMS objects.
 * @exception JMSException if JMS fails to close objects due to internal error
 */
private void close() throws JMSException {
    qsender.close();
Example 44-4  Collection Model Returned by Tree Binding

package ads.demo.view;
import ads.demo.common.DemoDataChangeEntry;
import ads.demo.common.DemoDataUpdateEvent;
import ads.demo.common.JMSHelper;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageListener;
import javax.jms.ObjectMessage;
import javax.jms.QueueReceiver;
import oracle.adf.model.binding.DCBindingContainer;
import oracle.adf.model.binding.DCIteratorBinding;
import oracle.adf.share.ADFContext;
import oracle.adf.view.rich.event.ActiveDataEntry;
import oracle.adf.view.rich.event.ActiveDataListener;
import oracle.adf.view.rich.event.ActiveDataUpdateEvent;
import oracle.adf.view.rich.model.ActiveDataModel;
import oracle.adfinternal.view.faces.model.binding.FacesCtrlHierBinding;
import oracle.jbo.Key;
import org.apache.myfaces.trinidad.model.CollectionModel;
...

public class ActiveDataCollectionModel extends CollectionModel implements ActiveDataModel,
...
    public void startActiveData(Collection<Object> rowKeys,
        int startChangeCount,
        ActiveDataListener listener) {
            _listeners.add(listener);
            _currEventId = startChangeCount;
            if (_listeners.size() == 1) { 
                // register as receiver for JMS Queue, listening to change event
                try {
                    String messageFilter = "JMSCorrelationID = " + getUuid() + ";";
                    qreceiver = JMSHelper.getInstance().createQueueReceiver(this, messageFilter);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
    }

    public void stopActiveData(Collection<Object> rowKeys,
        ActiveDataListener listener) {
            _listeners.remove(listener);
            if (_listeners.isEmpty()) { 
                // clean JMS
                try {
                    qreceiver.close();
                } catch (JMSException e) {
                    e.printStackTrace();
                }
            }
        }
public int getCurrentChangeCount() {
    return _currEventId;
}

public void onMessage(Message message) {
    try {
        DemoDataUpdateEvent myEvent = null;
        if (message instanceof ObjectMessage) {
            myEvent = (DemoDataUpdateEvent)((ObjectMessage)message).getObject();
            // Convert the event to ADS DataChangeEvent
        }
        List<ActiveDataEntry> dces = new ArrayList<ActiveDataEntry>(1);
        for (DemoDataChangeEntry entry : myEvent.getEntries()) {
            oracle.jbo.Key jboKey = new Key(entry.getPk());
            ActiveDataEntry.ChangeType newType = convertChangeType(entry.getType());
            Object[] path = new Object[] { Collections.singletonList(jboKey) };
            ActiveDataEntry dce =
                new DemoActiveDataEntry(newType, path,
                new Object[0],
                entry.getAttributes(),
                entry.getValues());
            dces.add(dce);
        }
        _currEventId++;
        ActiveDataUpdateEvent event = new DemoActiveDataUpdateEvent(new Object(), _currEventId,
            dces);
        _refreshControl = true;
        // Deliver event
        for (ActiveDataListener listener : _listeners) {
            try {
                listener.dataChanged(event);
            } catch (Throwable e) {
                e.printStackTrace();
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private int _currEventId = 0;
private final List<ActiveDataListener> _listeners =
    new LinkedList<ActiveDataListener>();
private boolean _refreshControl = false;
private String _treeBindingName;
private String _iterBindingName;
private ActiveDataEntry.ChangeType convertChangeType(DemoDataChangeEntry.ChangeType type) {
    if (type == DemoDataChangeEntry.ChangeType.UPDATE) {
        return ActiveDataEntry.ChangeType.UPDATE;
    } else if (type == DemoDataChangeEntry.ChangeType.REFRESH) {
        return ActiveDataEntry.ChangeType.REFRESH;
    } else {
        return ActiveDataEntry.ChangeType.UPDATE;
    }
    // Return ActiveDataEntry.ChangeType.UPDATE;
}

private CollectionModel getModel() {
    CollectionModel cm =
        (CollectionModel)ADFContext.getCurrent().getRequestScope().get("collectionModel_" +
```
this.hashCode());

DCBindingContainer bindings =
   (DCBindingContainer)ADFContext.getCurrent().getRequestScope().get("bindings");
if (_refreshControl) {
   DCIteratorBinding iterBinding =
      bindings.findIteratorBinding(_iterBindingName);
   iterBinding.executeQuery();
   _refreshControl = false;
}
if (cm == null) {
   FacesCtrlHierBinding hierBinding =
      (FacesCtrlHierBinding)bindings.findCtrlBinding(_treeBindingName);
   cm = hierBinding.getCollectionModel();
   ADFContext.getCurrent().getRequestScope().put("collectionModel_" +
      this.hashCode(), cm);
   System.out.println("CollectionModel: " + cm.hashCode());
}
return cm;
}
```

**Example 44-5  From the ActiveDataCollectionHandler Code**

```java
private CollectionModel getModel() {
   CollectionModel cm =
      (CollectionModel)ADFContext.getCurrent().getRequestScope().get("collectionModel_" +
      this.hashCode());

   DCBindingContainer bindings =
      (DCBindingContainer)ADFContext.getCurrent().getRequestScope().get("bindings");
   if (_refreshControl) {
      DCIteratorBinding iterBinding =
         bindings.findIteratorBinding(_iterBindingName);
      iterBinding.executeQuery();
      _refreshControl = false;
   }
   if (cm == null) {
      FacesCtrlHierBinding hierBinding =
         (FacesCtrlHierBinding)bindings.findCtrlBinding(_treeBindingName);
      cm = hierBinding.getCollectionModel();
      ADFContext.getCurrent().getRequestScope().put("collectionModel_" +
         this.hashCode(), cm);
   }
   return cm;
}
```

### 44.4.2 Building the Supporting Active Data Entry Classes

The [ActiveDataCollectionHandler](#) uses Oracle ADF Rich Events to propagate the data updates and UI refresh in response to JMS queue updates. You must implement these event classes and register them as events from the [CollectionHandler](#).

**To create the Active Data Entry implementation:**

The class shown in **Example 44-6** extends the Oracle ADF class `oracle.adf.view.rich.event.ActiveDataEntry` and implements several methods in that interface.

**To implement the Active Data Update Event:**

The Active Data update event takes a list of Active Data entry events and performs them at once. The class extends from
oracle.adf.view.rich.event.ActiveDataUpdateEvent and implements several methods, as shown in Example 44-7.

**Example 44-6  Active Data Entry Class**

```java
package ads.demo.view;

import java.util.HashMap;
import java.util.Map;
import oracle.adf.view.rich.event.ActiveDataEntry;

public class DemoActiveDataEntry extends ActiveDataEntry {
    public DemoActiveDataEntry(ActiveDataEntry.ChangeType change,
            Object[] path, Object[] insertKeyPath,
            String[] names, Object[] values) {
        super();

        if (names != null) {
            for (int i = 0; i < names.length; i++) {
                String attribute = names[i];
                Object value = values[i];
                _valuesMap.put(attribute, value);
            }
        }

        _attributes = names;
        _values = values;
        _changeType = change;
        _path = path;
        _insertPath = insertKeyPath;
    }

    public ActiveDataEntry.ChangeType getChangeType() {
        return _changeType;
    }

    public Object[] getKeyPath() {
        return _path;
    }

    public Object[] getInsertKeyPath() {
        return _insertPath;
    }

    public String[] getAttributeNames() {
        return _attributes;
    }

    public Object getAttributeValue(String name) {
        return _valuesMap.get(name);
    }

    public Object getFormattedAttributeValue(String name) {
        return getAttributeValue(name);
    }

    private final Map<String, Object> _valuesMap =
            new HashMap<String, Object>();
```
private String[] _attributes = null;
private Object[] _values = null;
private Change _changeType = null;
private Object[] _path = null;
private Object[] _insertPath = null;
}

Example 44-7   Active Data Update Event

package ads.demo.view;
import java.util.Collections;
import java.util.List;
import oracle.adf.view.rich.event.ActiveDataEntry;
import oracle.adf.view.rich.event.ActiveDataUpdateEvent;

public class DemoActiveDataUpdateEvent extends ActiveDataUpdateEvent {
    public DemoActiveDataUpdateEvent(Object object) {
        super(object);
    }

    public DemoActiveDataUpdateEvent(Object source, int eventId,
            List<ActiveDataEntry> changeList) {
        super(source);
        _changeList = changeList;
        _eventId = eventId;
    }

    /**
     * Get the change list of this event
     * @return the change list of this event
     */
    public List<ActiveDataEntry> getChangeList() {
        return _changeList;
    }

    /**
     * Get the event ID
     * @return the event ID
     */
    public int getEventId() {
        return _eventId;
    }

    public long getEventTime() {
        return System.currentTimeMillis();
    }

    public String toString() {
        return super.toString() + " eventId:" + _eventId + " changeList:" +
            _changeList;
    }
}
44.4.3 Register the Active Data Collection Model with the Oracle ADF UI Page

To enable the active data feature and "hook" your collection model, you need to register the class as a managed JavaBean.

ADS requires UI components to have the same model across requests. Therefore, register the `ActiveDataCollectionModel` as a view scoped managed JavaBean. If you stay on the same page, the table is based on the same model.

To register your collection model as a managed JavaBean:

1. Open `adfc-config.xml`.
2. Add the managed JavaBean named `adsBean` and provide the package to your collection model class, as shown in Example 44-8.

**Example 44-8   adsBean Managed JavaBean**

```xml
<managed-bean>
  <managed-bean-name>adsBean</managed-bean-name>
  <managed-bean-class>ads.demo.view.ActiveDataCollectionModel</managed-bean-class>
  <managed-bean-scope>view</managed-bean-scope>
  <managed-property>
    <property-name>treeBindingName</property-name>
    <property-class>java.lang.String</property-class>
    <value>EmpView1</value>
  </managed-property>
  <managed-property>
    <property-name>iterBindingName</property-name>
    <property-class>java.lang.String</property-class>
    <value>EmpView1Iterator</value>
  </managed-property>
</managed-bean>
```

44.4.4 Register the Component Managed JavaBean for Supporting Method Actions

To trigger the synchronous functionality of the use case pattern, raise a business event in response to the click of an Oracle ADF button. To support a response to the click of a button, create a managed JavaBean with which you can associate methods as the action for these buttons.

To build your Oracle ADF component managed JavaBean:

In the prototype use case, there is a table that contains a list of employees and their entity object attributes. Add two buttons at the top of the table in a panel collection toolbar which, when clicked, uses the selected employee to initiate an approval process. When completed, the approval process dynamically updates the table, as shown in Figure 44-2.
The table component requires the managed JavaBean shown in Example 44-9.

To register the component managed JavaBean:
As with the collection model, register the component managed JavaBean by adding an entry to adfc-config.xml, as shown in Example 44-10.

**Example 44-9 Table Component Managed JavaBean**

```java
package ads.demo.view;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Map;
import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
import javax.faces.event.ActionEvent;
import oracle.adf.model.OperationBinding;
import oracle.adf.model.binding.DCBindingContainer;
import oracle.adf.share.ADFContext;
import oracle.adf.view.rich.component.rich.data.RichTable;
import oracle.jbo.Key;
import org.apache.myfaces.trinidad.model.RowKeySet;

public class TableHandlerBean {
    private RichTable _table;

    public TableHandlerBean() {
        super();
    }

    public void setTable(RichTable _table) {
        this._table = _table;
    }
}
```
public RichTable getTable() {
    return _table;
}

public void handleRaise(ActionEvent event) {
    String correlationId = ((ActiveDataCollectionModel)ADFContext.getCurrent().getViewScope().get("adsBean")).getUuid();
    RowKeySet selectedRowKeys = getTable().getSelectedRowKeys();
    ArrayList<String> selectedEmp = new ArrayList<String>(selectedRowKeys.size());
    for (Object rowKey : selectedRowKeys) {
        Key jboKey = ((Collection<Key>)rowKey).iterator().next();
        String rowKeyString = ((Integer)jboKey.getKeyValues()[0]).toString();
        selectedEmp.add(rowKeyString);

        // Publish event
        try {
            DCBindingContainer bindings = (DCBindingContainer)ADFContext.getCurrent().getRequestScope().get("bindings");
            OperationBinding action = (OperationBinding)bindings.findCtrlBinding("publishEvent");
            Map params = action.getParamsMap();
            params.put("correlationId", correlationId);
            params.put("key", rowKeyString);
            params.put("eventType", "payRaise");
            action.execute();
        } catch ( Exception e ) {
            log.severe("ASM: Failed to raise commission event for key: " + rowKeyString);
        }
    }
}

// Invoke BPEL from here.

public void handleCommission(ActionEvent event) {
    String correlationId = ((ActiveDataCollectionModel)ADFContext.getCurrent().getViewScope().get("adsBean")).getUuid();
    RowKeySet selectedRowKeys = getTable().getSelectedRowKeys();
    ArrayList<String> selectedEmp = new ArrayList<String>(selectedRowKeys.size());
    for (Object rowKey : selectedRowKeys) {
        Key jboKey = ((Collection<Key>)rowKey).iterator().next();
        String rowKeyString = ((Integer)jboKey.getKeyValues()[0]).toString();
        selectedEmp.add(rowKeyString);

        // Publish event
        try {
            DCBindingContainer bindings = (DCBindingContainer)ADFContext.getCurrent().getRequestScope().get("bindings");
            OperationBinding action = (OperationBinding)bindings.findCtrlBinding("publishEvent");
            Map params = action.getParamsMap();
            params.put("correlationId", correlationId);
            params.put("key", rowKeyString);
            params.put("eventType", "payCommission");
            action.execute();
        } catch ( Exception e ) {
            log.severe("ASM: Failed to raise commission event for key: " + rowKeyString);
        }
    }
}

// Invoke BPEL from here.
private void addConfirmationMessage() {
    FacesMessage msg = new FacesMessage("You request is submitted for approval.");
    FacesContext.getCurrentInstance().addMessage(null, msg);
}

Example 44-10  adfc-config.xml Registration Code

<managed-bean>
    <managed-bean-name>tableBean</managed-bean-name>
    <managed-bean-class>ads.demo.view.TableHandlerBean</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>

44.4.5 Reference the Managed JavaBean in the Page UI

Modify the page component to reference the managed JavaBean from the earlier steps, as shown in Example 44-11.

Note:
You may notice that selectedRowKeys is not bound to any method. By default, it is bound to #{bindings.treeBinding.collectionModel.selectedRowKeys}. It will no longer work after using ActiveDataCollectionModel.

Example 44-11  Referencing the Managed JavaBean

<af:table value="#{viewScope.adsBean}" var="row"
    rows="#{bindings.EmpView1.rangeSize}"
    fetchSize="#{bindings.EmpView1.rangeSize}"
    rowBandingInterval="0"
    filterModel="#{bindings.EmpView1Query.queryDescriptor}"
    queryListener="#{bindings.EmpView1Query.processQuery}"
    filterVisible="true" varStatus="vs"
    selectionListener="#{bindings.EmpView1.collectionModel.makeCurrent}"
    rowSelection="multiple" id="t1" width="100%"
    binding="#{tableBean.table}"

44.4.6 Create the Data Model and Add Application Module Methods

The data model should exist before the page is built to simplify laying out the components required to display the data contained in that model. The application module needs additional methods to support incoming service methods and, optionally, the methods for raising the business event.

To extend the methods of your application module for the service interface:

Make sure to expose one or more application module methods in the application module service. This facilitates the callback from BPEL upon completion of the process, triggering the ADS UI update. These methods publish the message to the JMS queue following the message structure shown here.

The message payload should take the format of DataUpdateEvent, which comprises one or more DataChangeEntry items.
• changeType: enum (UPDATE, REFRESH). Currently, there is no use case for INSERT.

• key: Object[]

• insertKey: Object[]

• attributeNames: String[], a list of names of changed attributes

• attributeValues: Object[], a list of new values for the changed attributes

In this pattern, payRaise and payCommission are supported for one or more selected employees. Use methods with simple string interfaces invoked by BPEL to complete the payRaise or payCommission event for each particular employee. Call the sendMessage method to publish the JMS message to notify ADS of the UI update. Sample BPEL methods are shown in Example 44-12.

To define structure and compose event metadata:

The code that programmatically creates business event payloads and raises them through the business event APIs should be deliberately built around the namespace and event attributes defined in the appropriate EDL and XSD files.

For this pattern, a single event is used that supports multiple event types through an attribute value such as payRaise and payCommission. However, support for additional event types only requires adding the UI facet, the programmatic method to raise that new event type and a conditional branch in BPEL. If the pattern requires completely separate event definitions, the code becomes more complex, the number of managed metadata source files increases, and the composite becomes more complex as well.

While this is a simpler approach, it is not as flexible from an integration perspective. Define your event types such that they support your current use case and potentially support additional integration in the future. Example 44-13 shows a simplified event definition, while Example 44-14 shows an event schema definition.

To extend the application module with publishEvent and supporting methods:

In the page bindings, add the method publishEvent that binds to the application module method of the same name. Use this binding in the handleRaise and handleCommission methods of the TableHandlerBean to publish the event for each employee to be updated.

For more information about extending the application module with the publishEvent method, see Using the Java Event API to Publish Events.

Note:

It is critical that the event name and namespace are consistent throughout the code and metadata definitions in the subscribing SOA composite.

Related Links

For more information about creating a data model with application modules, see the chapter “Implementing Business Services with Application Modules” in Developing Fusion Web Applications with Oracle Application Development Framework.
Example 44-12  BPEL Methods

// Simplified interface method for service call per employee

    public void performSingleRaise(String correlationId, String key) {
        ArrayList thelist = new ArrayList<String>();
        thelist.add(key);
        performRaise(correlationId, thelist);
    }

// List interface for call from UI and by Simplified Service Method

    public void performRaise(String correlationId, List<String> keyValues) {
        List<DemoDataChangeEntry> dces = new ArrayList<DemoDataChangeEntry>(keyValues.size());
        ViewObject empVO = getEmpView1();
        for (String keyVal : keyValues) {
            Key key = new Key(new Object[] { keyVal });
            Row row = empVO.findByKey(key, 1)[0];
            BigDecimal newSal = new BigDecimal(Math.round(((BigDecimal)row.getAttribute("Sal")).doubleValue()*(1+(new Random()).nextDouble()/10)));
            row.setAttribute("Sal", newSal);
            DemoDataChangeEntry dce = new DemoDataChangeEntry(new Object[] { new Integer(keyVal) }, DemoDataChangeEntry.ChangeType.UPDATE, new String[] { "Sal" }, new Object[] { newSal.toString() });
            dces.add(dce);
            this.getDBTransaction().commit();
            DemoDataUpdateEvent event = new DemoDataUpdateEvent(dces);
            // Send a message
            sendMessage(correlationId, event);
        }
    }

// Simplified interface for Service method

    public void paySingleCommission(String correlationId, String key) {
        ArrayList<String> thelist = new ArrayList<String>();
        thelist.add(key);
        payCommission(correlationId, thelist);
    }

// List interface for calling from UI and by Simplified Service Method

    public void payCommission(String correlationId, List<String> keyValues) {
        List<DemoDataChangeEntry> dces = new ArrayList<DemoDataChangeEntry>(keyValues.size());
        ViewObject empVO = getEmpView1();
        for (String keyVal : keyValues) {
            Key key = new Key(new Object[] { keyVal });
            Row row = empVO.findByKey(key, 1)[0];
            BigDecimal newComm = new BigDecimal((new Random()).nextInt(10000));
            row.setAttribute("Comm", newComm);
            DemoDataChangeEntry dce = new DemoDataChangeEntry(new Object[] { new Integer(keyVal) }, DemoDataChangeEntry.ChangeType.REFRESH,
new String[] { "Comm" },
new Object[] { newComm.toString() });

dces.add(dce);
} this.getDBTransaction().commit();

DemoDataUpdateEvent event = new DemoDataUpdateEvent(dces);
// send message
sendMessage(correlationId, event);
}

// Private method to push ADS update to JMS queue
private void sendMessage(String correlationId, DemoDataUpdateEvent event) {
    try {
        JMSHelper helper = JMSHelper.getInstance();
        ObjectMessage message = helper.createObjectMessage();
        message.setObject(event);
        message.setJMSCorrelationID(correlationId);
        helper.sendMessage(message);
    } catch (JMSException e) { e.printStackTrace();
    } // try
} // sendMessage

**Example 44-13 Simplified Event Definition**

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions xmlns="http://schemas.oracle.com/events/edl"
    targetNamespace="http://xmlns.oracle.com/apps/ta/adsdemo/events/edl">
    <schema-import namespace="http://xmlns.oracle.com/apps/ta/adsdemo/events/schema"
        location="xsd/ADSDemoEventSchema.xsd"/>
    <event-definition name="ADSDemoEvent">
        <content xmlns:ns0="http://xmlns.oracle.com/apps/ta/adsdemo/events/schema"
            element="ns0:ADSDemoEventElement"/>
    </event-definition>
</definitions>
```

**Example 44-14 Event Schema Definition**

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://xmlns.oracle.com/apps/ta/adsdemo/events/schema"
    targetNamespace="http://xmlns.oracle.com/apps/ta/adsdemo/events/schema"
    attributeFormDefault="unqualified"
    elementFormDefault="qualified">
    <xsd:element name="ADSDemoEventElement" type="ADSDemoEventElementType"/>
    <xsd:complexType name="ADSDemoEventElementType">
        <xsd:sequence>
            <xsd:element name="correlationId" type="xsd:long"/>
            <xsd:element name="key" type="xsd:long"/>
            <xsd:element name="eventType" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```
44.4.7 Create a SOA Composite that Subscribes to the Published Event

The creation of a SOA composite that subscribes to an event is covered in \textit{Initiating a SOA Composite from an Web Application}. A sample pattern composite is shown in Figure 44-3.

\textit{Figure 44-3} \hspace{1em} \textit{Pattern Composite}

\begin{center}
\includegraphics[width=\textwidth]{pattern_composite.png}
\end{center}

44.4.8 Construct a BPEL Process to Perform Asynchronous Work

The creation of a BPEL process and human task activities is described in other sections. For more information, see \textit{Managing Tasks from an Application}. A sample BPEL process is shown in Figure 44-4.

\begin{center}
\includegraphics[width=\textwidth]{bpel_process.png}
\end{center}
44.4.9 Invoking the ADF Business Components Service

Invoking an ADF Business Components service from a BPEL process is covered in another section. For more information, see Orchestrating ADF Business Components Services.

44.5 Securing the Design Pattern

The process of securing this design pattern is the same as that of securing an Oracle ADF UI application.

For more information, see Securing Web Services Use Cases.

44.6 Verifying the Deployment

Do the following to test functionality:

1. Turn on the EDN-DB-LOG page by navigating to http://host:port/soa-infra/events/edn-db-log and ensure it reads "Log is Enabled." If it is not, click Enable.

2. Open the UI page and interact with the UI components that you designed to trigger the event.
   The event should immediately display in the EDN-DB-LOG page.

3. Check for the event payload shown in Example 44-15.

4. Check the console ($DOMAIN_HOME/as.log) or soa-diagnostic logs ($DOMAIN_HOME/servers/<servername>logs/<servername>.log) to see any Mediator activity that results from your event.
INFO: MediatorServiceEngine received an event = {http://xmlns.oracle.com/apps/ta/adsdemo/events/edl}ADSDemoEvent
Apr 17, 2009 1:57:26 PM oracle.tip.mediator.common.persistence.MediatorPersistor
persistCallback
INFO: No call back info set in incoming message
Apr 17, 2009 1:57:26 PM oracle.tip.mediator.common.persistence.MediatorPersistor
persistCallback
INFO: Message properties :{id=041ecfcf-8b73-4055-b5c0-0b89af04f425,
tracking.compositeInstanceId=50003,
tracking.ecid=000012pgzVCBLA5xrO17SY19uEYF00004g:47979}
Apr 17, 2009 1:57:26 PM oracle.tip.mediator.dispatch.InitialMessageDispatcher
dispatch
INFO: Executing Routing Service..
Apr 17, 2009 1:57:26 PM oracle.tip.mediator.dispatch.InitialMessageDispatcher
processCases
INFO: Unfiltered case list size :1
Apr 17, 2009 1:57:26 PM oracle.tip.mediator.monitor.MediatorActivityMonitor
createMediatorCaseInstance
INFO: Creating case instance with
name :ADEDemoProcess.ademoprocess_client.process
Apr 17, 2009 1:57:26 PM oracle.tip.mediator.dispatch.InitialMessageDispatcher
processCase
INFO: Immediate case {ADEDemoProcess.ademoprocess_client.process} with case
id :{(5852BA028921DEAF6403EF6E2FB21D) will be executed
Apr 17, 2009 1:57:26 PM oracle.tip.mediator.service.filter.FilterFactory
createFilterHandler
INFO: No Condition defined

5. Check Oracle Enterprise Manager at http://host:port/em for an instance
of your SOA composite, and check for errors.

6. If your process has no errors and is expecting a response from the human
workflow notification, do the following:
   a. Navigate to the worklist at http://host:port/integration/
      worklistapp.
   b. Log in as the assigned approver.
   c. Approve or reject the notification per your design requirements.

At this point, the BPEL process should complete and invoke the ADF Business
Components service to trigger the ADS push. The UI should promptly update. Check
the Oracle ADF UI runtime console and diagnostic logs for stack traces and log
messages.

**Example 44-15  Event Payload**

Enqueing event: http://xmlns.oracle.com/apps/ta/adsdemo/events/edl:ADSDemoEvent from J
xmlns="http://oracle.com/fabric/businessEvent">
  <name>ns:ADSDemoEvent</name>
  <id>494ae921-4667-4a42-8190-5a5aaa428f7e</id>
  <content>
    <ADSDemoEventElement xmlns="http://xmlns.oracle.com/apps/ta/adsdemo/events/schema">
      <correlationId>3926ed2d-e023-4f05-85f9-bdf0b57099ae</correlationId>
      <key>7499</key>
      <eventType>payRaise</eventType>
    </ADSDemoEventElement>
  </content>
</business-event>
Subject name:
Enqueing complete

Starting EDN Agent for Event from Queue
Dequeued event: http://xmlns.oracle.com/apps/ta/adsdemo/events/edl::ADSDemoEvent

Subject name:
  <name>ns:ADSDemoEvent</name>
  <id>494ae921-4667-4a42-8190-5a5aaa428f7e</id>
  <content>
    <ADSDemoEventElement xmlns="http://xmlns.oracle.com/apps/ta/adsdemo/events/schema">
      <correlationId>3926ed2d-e023-4f05-85f9-bdf0b57099ae</correlationId>
      <key>7499</key>
      <eventType>payRaise</eventType>
    </ADSDemoEventElement>
  </content>
</business-event>

44.7 Troubleshooting the Use Case

For the Oracle ADF UI functionality, use Fusion Middleware Control, Oracle Fusion Applications Logger, and server diagnostic logs for information about what is failing.

For the events functionality, use the Event Delivery Network database log page at http://host:port/soa-infra/events/edn-db-log.

For the SOA functionality, use the Oracle Enterprise Manager console for diagnostics and Oracle Fusion Applications Logger sensor variables for logging.

For the ADF Business Components service functionality, use BPEL fault handling and logging via Oracle Fusion Applications Logger sensor variables as well as the console, Oracle Fusion Applications Logger and server diagnostic logs for more detailed error messages.

44.8 What You May Need to Know About Initiating an Asynchronous Service with Dynamic UI Update

- Oracle ADF UI
- ADS
- JMS
- Business Events (programmatic)
- SOA Mediator
- SOA BPEL
- ADF Business Components services

44.9 Known Issues and Workarounds

Known issues are as follows:

- Sparkle does not occur on selected rows.
Manage Tasks Programmatically

This chapter describes what to do when you need to programmatically create, set an outcome for, or query task information that resides in one or more SOA domains.

**When to implement:** When you need to programmatically create, set an outcome for, or query task information that resides in one or more SOA domains.

**Design Pattern Summary:** The design pattern involves programmatic interaction with human task client services by an application with the following requirements: displaying task status information, providing UI facets to enable setting task outcomes without navigating through the worklist, and submitting new tasks without initiating a BPEL process.

In addition, the human task services support federated task queries across several SOA domains so as to obtain an aggregated task list across several product families.

**Involved components:**

- Java code, such as an Oracle ADF application module or Oracle Enterprise Scheduler Java job
- SOA Domain with deployed human task

**45.1 Introduction to the Recommended Design Pattern**

Some Oracle Fusion web applications have use cases that require programmatically interacting with the human workflow layer to approve, reject and display lists of tasks using specific search criteria. All SOA runtime environments that are configured as part of the topology are stored in the Oracle Fusion Middleware Extensions for Applications taxonomy schema and accessed at runtime using APIs. In the taxonomy schema, each SOA runtime environment has an entry with a unique identifying name that maps to a corresponding endpoint URL. For example, a SOA runtime environment called `FIN_SOA_RMI` has a corresponding endpoint URL `t3://fpp-02.mycompany.com:7001/`.

At runtime, the taxonomy schema is queried to construct a list of servers and their respective endpoints. This list of servers and endpoints passes to the human workflow client service APIs as a JAXB object. In the context of the federated task query service, some or all of these servers can be referenced. One of these servers is set as the default, and is used in the context of non-federated task services such as the task and task query services.

Alternatively, servers can be excluded from the list of federated servers, and exist only in the JAXB object. This allows servers to be used only when named explicitly in the list of requested servers. In this case, the excluded servers will not be used when the list of requested servers is empty.
Oracle Fusion Middleware Extensions for Applications maintains the list of servers in the taxonomy tables. An API enables building the JAXB object based on the list of SOA domains in the Oracle Fusion Applications topology.

This pattern is recommended as it provides the following features:

- Federated query support,
- Programmatic access supports custom UI requirements and ADF Business Components services and Oracle Enterprise Scheduler job integration.

### 45.2 Potential Approaches

- Supported approach: Managing Tasks from an Oracle ADF application.
- Unsupported approach: Direct invoking a service using JAX-WS proxies.

### 45.3 Example

The Expenses team has an Expenses Manager role with administrator privileges to approve or reject expenses that belong to other users. The Expenses team must provide a workbench that collectively scans all SOA domains for open expense notifications and provide a consolidated UI to set their outcome, potentially all at once. This UI would comprise a table listing notifications matching certain filter criteria with buttons to select and set the appropriate outcome of the expense. This is done through RMI interaction with the appropriate SOA domain.

### 45.4 Manage Human Workflow Tasks from a Java Application

These are two main high-level steps involved in this process:

- Create and deploy a human task definition using a SOA composite that contains the human task.
- Develop code to do the following:
  - Connect to the task services.
  - Query or lookup the tasks.
  - Display the task summary.
  - Set the desired task outcome.

#### 45.4.1 How to Connect to the Task Service/Task Query Service

The human workflow APIs provide three types of task services: single, query and federated query. The type of service you use depends on the product use case. The RMI endpoint for these services must be derived at runtime and compiled into a server list. The server list is contained by a JAXB object, which can be passed to the human workflow client service APIs. To support this runtime lookup, the Oracle Fusion Middleware Extensions for Applications taxonomy schema and APIs must be seeded during provisioning. You need only provide an `ArrayList` (`java.util.ArrayList`) of server names to be used in the federated query.
Note:
Consider performance requirements when using the query or federated query service APIs. It is recommended to page the result sets in batches, for example, in sets of 10-25.

Services are as follows:

- **Task Service**: The task service programmatically sets a task outcome, such as approve or reject for a single, particular task using a single, particular SOA runtime.

- **Task Query Service**: The task query service programmatically queries tasks for a particular task type. Use the portlet to render a worklist and relevant tasks in your dashboard. Alternatively, you may manually build the worklist instead.

- **Federated Task Query Service**: The federated task query service programmatically executes a federated query of tasks for a particular task type. Use the portlet to render a worklist and relevant tasks in your dashboard. Alternatively, you may manually build the worklist instead.

Use the following guidelines to determine the type of task or query service to use.

- **Single server task service API**: Use this API to obtain a single task object using the task number or task ID from a single, specific SOA runtime.

- **Single server task query service API**: Use this API to query for tasks from a single, specific SOA runtime.

- **Federated task query service API**: Use this API to query tasks based on namespace or task name from one or more SOA runtime domains.

### 45.4.2 How to Use the Single Server Task Service API

If your use case requires connecting to one SOA domain and obtaining the details of a single task via a primary key such as task number or task ID, take the following steps. After it is obtained, configure the task detail display or set the task outcome.

- Import libraries into the Java project.
- Import code packages into the Java project.
- Declare and obtain task service object references.

#### 45.4.2.1 Import Libraries into the Java Project

Add the following libraries to the Oracle JDeveloper project:

- **Applications Core**: Add this library to enable using Oracle Fusion Middleware Extensions for Applications taxonomy APIs to obtain the necessary JAXB object containing the RMI endpoint information for the desired server.
- **SOA Runtime**: Add this library to enable using the human workflow task query APIs.

#### 45.4.2.2 Import Code Packages into the Java Project

Import the code packages shown in Example 45-1 into the Java source.
**Example 45-1 Importing Code Packages to Enable Using the Single Server Task Service API**

```java
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Logger;
import javax.jws.WebService;
import oracle.bpel.services.workflow.verification.IWorkflowContext;
import oracle.bpel.services.workflow.client.IWorkflowServiceClient;
import oracle.bpel.services.workflow.task.model.Task;
import oracle.bpel.services.workflow.IWorkflowConstants;
import oracle.bpel.services.workflow.client.WorkflowServiceClientFactory;
import oracle.bpel.services.workflow.query.ITaskQueryService;
import oracle.bpel.services.workflow.repos.Ordering;
import oracle.bpel.services.workflow.repos.Predicate;
import oracle.bpel.services.workflow.repos.TableConstants;
import oracle.bpel.services.workflow.task.IInitiateTaskResponse;
import oracle.bpel.services.workflow.task.ITaskService;
import oracle.bpel.services.workflow.task.model.ObjectFactory;
import oracle.apps.fnd.applcore.common.DeploymentsUtil;
import oracle.bpel.services.workflow.client.config.RemoteClientType;
import oracle.bpel.services.workflow.client.config.ServerType;
import oracle.bpel.services.workflow.client.config.WorkflowServicesClientConfigurationType;
```

45.4.2.3 Declare and Obtain Task Service Object References

Create the query and task service references by invoking the `WorkflowServiceClientFactory` API `getWorkflowServiceClient` method, which provides the JAXB object containing the server list and a logger object reference. The human workflow task service connects to the server marked as the default in the JAXB object. When calling the APIs to craft the JAXB object, be sure to specify the name of the server you want to call. Example 45-2 shows sample code used to declare and obtain task service object references.

**Note:**

Previously, developers would populate the properties for `EJB_PROVIDER` and `EJB_SECURITY` to provide the RMI endpoint and credentials. Instead, RMI identity propagation uses the current user context for authentication. In Oracle Fusion Applications, most UI and services require authentication that provides the appropriate user context. If no current user context exists, create one.

**Example 45-2 Declaring and Obtaining Task Service Object References**

```java
ITaskService taskSvc = null;
ITaskQueryService querySvc = null;
IWorkflowContext wfCtx = null;
java.util.logging.Logger logger = Logger.getLogger("oracle.apps");
try {
    WorkflowServicesClientConfigurationType wscct =
    getWorkflowClientConfigObject("FIN_SOA_RMI");
    if ( wscct == null ) {  //Log incident
        return "FAILED!";
    }
```
} // if

IWorkflowServiceClient wfSvcClient = WorkflowServiceClientFactory.getWorkflowServiceClient(wscct, logger);
taskSvc = wfSvcClient.getTaskService();
querySvc = wfSvcClient.getTaskQueryService();

### 45.4.2.4 Obtain the Workflow Service Context Object

To sustain performance in all interactions to the workflow client service APIs, pass the workflow service context object to any applicable APIs. To obtain the context using the current user's identity, call the `getWorkflowContextForAuthenticatedUser()` method, as shown in Example 45-3.

**Note:**

For performance reasons, be sure to pass this context to all subsequent calls to the APIs.

---

**Example 45-3   Getting the Workflow Service Context Object**

```java
// Get the workflow task service context for use in later calls for performance improvement
wfCtx = querySvc.getWorkflowContextForAuthenticatedUser();
```

### 45.4.2.5 Obtain the Single Task Object and Set Task Outcome

When interacting with the task service, you must first obtain the task number or ID for the task to retrieve the task details. Use the task number or task ID to invoke the `getTaskDetailsById` or `getTaskDetailsByNumber` methods of the task query service object.

**Note:**

This approach assumes that you have obtained the task number or task ID (either through the task query service or otherwise).

---

**Example 45-4** shows the approval of a task with the ID 0a6d287a-9849-4e5e-914b-805706d6b9d9.

**Example 45-5** shows how to use STDOUT calls to display the various task attributes through the task API.

**Example 45-4   Getting the Single Task Object with the Task ID**

```java
Task t = querySvc.getTaskDetailsById(wfCtx, "0a6d287a-9849-4e5e-914b-805706d6b9d9");
taskSvc.updateTaskOutcome(wfCtx, t, "APPROVE");
// Another example, using the task number to reject a task.
Task t = querySvc.getTaskDetailsByNumber(wfCtx, 200140 );
taskSvc.updateTaskOutcome(wfCtx, t, "REJECT");
```

**Example 45-5   Using STDOUT Calls to Display Task Attributes**

```java
System.out.println("Task Number: " + task.getSystemAttributes().getTaskNumber());
System.out.println("Task Id: " + task.getSystemAttributes().getTaskId());
System.out.println("Title: " + task.getTitle());
System.out.println("Priority: " + task.getPriority());
System.out.println("State: " + task.getSystemAttributes().getState());
```
45.4.3 How to Use the Single Server Task Query Service API

If your use case involves connecting to a single SOA domain and querying for all tasks that match certain criteria, take the following steps. After you have queried for the relevant tasks, you can display them in an ordered list in the UI or programmatically set task outcome all at once.

- Import libraries into the Java project.
- Import code packages into the Java project.
- Declare and obtain task query service object references.
- Manage query and task outcome states.

45.4.3.1 Import Libraries into the Java Project

Import the libraries described in Import Libraries into the Java Project.

45.4.3.2 Import Code Packages into the Java Project

Import the code packages described in Import Code Packages into the Java Project.

45.4.3.3 Declare and Obtain Task Query Service Object References

Create the query and task service references by invoking the `WorkflowServiceClientFactory` API `getWorkflowServiceClient` method, which provides the JAXB object containing the server list and a logger object reference. The human workflow task service connects to the server marked as the default in the JAXB object. When calling the APIs to craft the JAXB object, be sure to specify the name of the server you want to call.

Example 45-6 shows sample code in which task query service object references are declared and obtained.

Note:

Previously, developers would populate the properties for `EJB_PROVIDER` and `EJB_SECURITY` to provide the RMI endpoint and credentials. Instead, RMI identity propagation uses the current user context for authentication. In Oracle Fusion Applications, most UI and services require authentication that provides the appropriate user context. If no current user context exists, create one.

---

Example 45-6 Declaring and Obtaining Task Query Service Object References

```java
ITaskService taskSvc = null;
ITaskQueryService querySvc = null;
IWorkflowContext wfCtx = null;
try {
    java.util.logging.Logger logger = Logger.getLogger("oracle.apps");
    WorkflowServicesClientConfigurationType wscct =
        getWorkflowClientConfigObject("FIN_SOA_RMI");
    if ( wscct == null )   {               // Log incident
        return "FAILED!";
    }
    // Log incident
    return "FAILED!";
}
```
IWorkflowServiceClient wfSvcClient = WorkflowServiceClientFactory.getWorkflowServiceClient(wscct, logger);
taskSvc = wfSvcClient.getTaskService();
querySvc = wfSvcClient.getTaskQueryService();

45.4.3.4 Manage Query and Task Outcome States
Performing queries and interacting with the task result set is similar for both the federated and non-federated task query services. For more information about this process, see How to Query and Traverse Federated and Non-federated Query Result Sets.

45.4.4 How to Use the Federated Server Task Query Service API
Using the federated server task query service API involves the following main steps:

- Import libraries into the Java project.
- Import code packages into the Java project.
- Create a list of servers for a parallel federated query.
- Declare task and query service references, and create the workflow client service object.
- Obtain the workflow service client.
- Implement exception handling for federated queries.
- Manage query and task outcome states.

45.4.4.1 Import Libraries into the Java Project
Import the libraries described in Import Libraries into the Java Project.

45.4.4.2 Import Code Packages into the Java Project
Import the code packages described in Import Code Packages into the Java Project.

In addition, import the code package shown in Example 45-7.

Example 45-7    Importing the Code Package IFederatedWorkflowContext
import oracle.bpel.services.workflow.fws.client.IFederatedWorkflowContext;

45.4.4.3 Create a List of Servers for a Parallel Federated Query
To leverage the federated query service, decide whether to query all human workflow services in Oracle Fusion Applications or just a subset of those services. The servers are named according to standards and are populated in a JAXB object which contain the service endpoints for lookup at runtime. You need only know the name or names of the product services you want to poll or provide a list.

To use a subset of the human workflow services, construct a Java list of those service names and pass that list to getFederatedTaskQueryService.
Note:

Be sure to provide a list of requested servers, as all servers in the list are polled. Failing to provide a list of servers results in all the servers being polled, which has significant performance implications.

Example 45-8 shows sample code in which a list of servers is created for a parallel federated query.

Example 45-8 Creating a List of Servers for a Parallel Federated Query

// Create the human workflow server subset list.
List<String> requestedServers = new ArrayList<String>();
requestedServers.add("FIN_SOA_RMI");
requestedServers.add("CRM_SOA_RMI");
requestedServers.add("PRJ_SOA_RMI");

45.4.4.4 Declare Task and Query Service References and Create the Workflow Client Service Object

After constructing the server list, obtain the query service object reference by invoking the `getFederatedTaskQueryService` API of the `WorkflowServiceClientFactory`, as shown in Example 45-9.

Note:

Previously, developers would populate the properties for `EJB_PROVIDER` and `EJB_SECURITY` to provide the RMI endpoint and credentials. Instead, RMI identity propagation uses the current user context for authentication. In Oracle Fusion Applications, most UI and services require authentication that provides the appropriate user context. If no current user context exists, create one.

Example 45-9 Declaring Task and Query Service References and Creating the Workflow Client Service Object

java.util.logging.Logger logger = Logger.getLogger("oracle.apps");
WorkflowServicesClientConfigurationType wscct =
    getWorkflowClientConfigObject("FIN_SOA_RMI");
if ( wscct == null ) {    // Log Incident
    return "FAILED!";
} // if
querySvc = WorkflowServiceClientFactory.getFederatedTaskQueryService(wscct,
    requestedServers, logger);

45.4.4.5 Obtain the Workflow Service Context

Obtain the workflow service context from the query service, as shown in Example 45-10. This improves performance with all workflow client service API interactions.

Note:

This context is cast as `IFederatedWorkflowContext`. For performance reasons, the context must be passed to all subsequent API calls.
45.4.4.6 Implement Exception Handling for Federated Queries

When performing queries on federated task query services, exceptions in communicating with any servers in the list of servers do not cause the query to fail. Instead, the context has a boolean isFailed() operation which can be interrogated to determine whether any failures occurred. Exceptions can be obtained from the context's getExceptionMap() method as shown in Example 45-11.

Example 45-11 Implementing Exception Handling

```java
// Partial success does not throw exceptions, instead check for isFailed and inspect the Exception and Context maps.
if (fedWFCtx.isFailed()) {
    // Log Messages
    logger.warning("Exception map: " + fedWFCtx.getExceptionMap());
    logger.warning("Context map: " + fedWFCtx.getWorkflowContextMap());
} // if
```

45.4.4.7 Manage Query and Task Outcome States

Performing queries and interacting with the task result set is similar for both the federated and non-federated task query services. For more information about this process, see How to Query and Traverse Federated and Non-federated Query Result Sets.

45.4.5 How to Query and Traverse Federated and Non-federated Query Result Sets

Querying and traversing federated and non-federated query result sets involves the following main steps:

- Determine query service search criteria.
- Construct the predicate for the queryTasks() method.
- Arrange the order of results returned by the queryTasks() method.
- Construct the list of display columns for the queryTasks() method.
- Construct a list of OptionalInfo items for the results of the queryTasks() method.
- Invoke the queryTasks() method with the attribute lists.
- Iterate through the result set.
- Programatically set the task outcome.

45.4.5.1 Determine Query Service Search Criteria

This section assumes you have implemented the task query service for either single or federated queries by following the instructions in How to Use the Single Server Task Query Service API and How to Use the Federated Server Task Query Service API.

For example, creating a human task in a composite produces a TASK file containing the metadata that defines the task behavior for approval hierarchy as well as possible outcomes. Examining the source of this TASK file reveals the task name, target namespace, possible outcomes, and so on. When deploying the composite containing
this task, the WFTASK tables are updated with task-related data that supports the human workflow infrastructure, as shown in Example 45-12.

The task metadata includes a value for targetNamespace, in this case http://xmlns.oracle.com/WFClientPatternSOAApp/WFClientPatternTaskComposite/Humantask1.

If you can access the SOAInfra schema of your runtime environment, you can query the details of this task with the query shown in Example 45-13.

This query results in the WFTASKMETADATA row, shown in Table 45-1 and Table 45-2.

### Table 45-1 WFTASKMETADATA Row Part One

<table>
<thead>
<tr>
<th>ID</th>
<th>URI</th>
<th>Name</th>
<th>Title</th>
<th>Component Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>default/SOAComposite1!1.0*2008-12-02_08-32-41_642/Humantask1</td>
<td>default/SOAComposite1!1.0*2008-12-02_08-32-41_642/Humantask1</td>
<td>Humantask1</td>
<td>string('Task Title')</td>
<td>Humantask1</td>
</tr>
</tbody>
</table>

### Table 45-2 WFTASKMETADATA Row Part Two

<table>
<thead>
<tr>
<th>CompositeDN</th>
<th>Composite Name</th>
<th>Composite Version</th>
<th>Namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>default/SOAComposite1!1.0*2008-12-02_08-32-41_642</td>
<td>SOAComposite</td>
<td>1.0</td>
<td><a href="http://xmlns.oracle.com/WFClientPatternSOAApp/WFClientPatternTaskComposite/Humantask1">http://xmlns.oracle.com/WFClientPatternSOAApp/WFClientPatternTaskComposite/Humantask1</a></td>
</tr>
</tbody>
</table>

You can then query the WFTASK table using the task ID shown in the first column of Table 45-1, as shown in Example 45-14.

The results of the entire table are too large to print, but the selected columns shown in Table 45-3 may be useful.

### Table 45-3 Useful Columns

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>ASSIGNED</td>
</tr>
<tr>
<td>TaskID</td>
<td>0a6d287a-9849-4e5e-914b-805706d6b9d9</td>
</tr>
<tr>
<td>TaskNumber</td>
<td>200140</td>
</tr>
<tr>
<td>WorkflowDescriptorURI</td>
<td>default/SOAComposite1!1.0*2008-12-02_08-32-41_642/Humantask1</td>
</tr>
<tr>
<td>TaskDefinitionID</td>
<td>default/SOAComposite1!1.0*2008-12-02_08-32-41_642/Humantask1</td>
</tr>
<tr>
<td>TaskDefinitionName</td>
<td>Humantask1</td>
</tr>
</tbody>
</table>
This example focuses on tasks belonging to the current user identity, obtained automatically from the session context and passed using a SAML token in the SOAP header. These tasks that bear the ASSIGNED state and match the namespace http://xmlns.oracle.com/WFClientPatternSOAApp/WFClientPatternTaskComposite/Humantask1.

Example 45-12  Sample `.task File Snippet`

```xml
<?xml version = '1.0' encoding = 'UTF-8'?>
<taskDefinition targetNamespace="http://xmlns.oracle.com/WFClientPatternSOAApp/WFClientPatternTaskComposite/Humantask1"
xmns:ora="http://schemas.oracle.com/xpath/extension"
xmns:task="http://xmlns.oracle.com/bpel/workflow/task"
xmns:xsd="http://www.w3.org/2001/XMLSchema"
xmns="http://xmlns.oracle.com/bpel/workflow/taskDefinition"
xmns:evidence="http://xmlns.oracle.com/bpel/workflow/TaskEvidenceService"
xmns:dvmm="http://www.oracle.com/XSL/Transform/java/oracle.tip.dvm.LookupValue"
xmns:ns0="http://xmlns.oracle.com/bpel/workflow/common"
xmns:hwf="http://xmlns.oracle.com/bpel/workflow/xpath"
xmns:tsc="http://xmlns.oracle.com/bpel/workflow/common/tsc"
xmns:xref="http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPathFunctions"
xmns:ids="http://xmlns.oracle.com/bpel/services/IdentityService/xpath"
  <name>Humantask1</name>
  ...
</taskDefinition>
```

Example 45-13  Query task details

```
select * from wftaskmetadata where namespace = 'http://xmlns.oracle.com/WFClientPatternSOAApp/WFClientPatternTaskComposite/Humantask1';
```

Example 45-14  Query the WFTASK table using the task ID

```sql
task:select * from wftask where taskdefinitionid = 'default/SOAComposite1!1.0*2008-12-02_08-32-41_642/Humantask1';
```

45.4.5.2 Construct the Predicate for queryTasks()

Predicate construction is the instantiation of objects which define an expression to model the conditional part of the underlying where clause. For example, a conditional statement such as TASKNUMBER = 200140 equates to the predicate constructor shown in Example 45-15.

For this case, a predicate is required to match the WFTASKMETA_NAMESPACE_COLUMN and the WFTASK_STATE_COLUMN columns to the appropriate values. First create the namespace column predicate, then the state column predicate, followed by a predicate combining these two predicates with a conditional, as shown in Example 45-16.
An additional way to construct a predicate is shown in Example 45-17, as specified in the ITaskQueryService documentation.

**Example 45-15  Constructing the Predicate**

```java
new Predicate(TableConstants.WFTASK_TASKNUMBER_COLUMN, Predicate.OP_EQ, 200140);
```

**Example 45-16  Creating Namespace, State and Combination Predicates**

```java
Predicate predicate1 = new Predicate(TableConstants.WFTASK_STATE_COLUMN, Predicate.OP_EQ, IWorkflowConstants.TASK_STATE_ASSIGNED);
Predicate predicate2 = new Predicate(TableConstants.WFTASKMETADATA_NAMESPACE_COLUMN, Predicate.OP_EQ, TASK_TARGET_NAMESPACE);
Predicate predicate = new Predicate(predicate1, Predicate.AND, predicate2);
```

**Example 45-17  Another Way to Construct a Predicate**

```java
Predicate statePredicate = new Predicate(TableConstants.WFTASK_STATE_COLUMN, Predicate.OP_NEQ, IWorkflowConstants.TASK_STATE_ASSIGNED);
statePredicate.addClause(Predicate.AND, TableConstants.WFTASK_NUMBERATTRIBUTE1_COLUMN, Predicate.OP_IS_NULL, nullParam);
Predicate datePredicate = new Predicate(TableConstants.WFTASK_ENDDATE_COLUMN, Predicate.OP_ON, new Date());
Predicate predicate = new Predicate(statePredicate, Predicate.AND, datePredicate);
```

**45.4.5.3 Arrange the Order of Results Returned by the queryTasks() Method**

This step is optional.

The Ordering parameter facilitates implementing an ORDER_BY clause in the task list query.

In Example 45-18, the TITLE_COLUMN and PRIORITY_COLUMN properties are added to the ORDER_BY clause.

**Example 45-18  Constructing the Ordering of queryTasks()**

```java
// Create the ordering
Ordering ordering = new Ordering(TableConstants.WFTASK_TITLE_COLUMN, true, true);
ordering.addClause(TableConstants.WFTASK_PRIORITY_COLUMN, true, true);
```

**45.4.5.4 Construct the List of Display Columns for the queryTasks() Method**

By default, the queryTasks() method returns only a list of tasks with their TASKID value. If you require additional columns, such as TASKNUMBER, TITLE, PRIORITY, STATE, ENDDATE, ASSIGNEE, COMPOSITEINSTANCEID, ROOTTASKID, and so on, construct an ArrayList of String objects containing the names of the additional columns you want returned in the result set. Example 45-19 shows sample code that constructs a list of display columns for queryTasks().

**Example 45-19  Constructing the List of Display Columns for queryTasks()**

```java
List queryColumns = new ArrayList();
queryColumns.add("TASKNUMBER");
```
queryColumns.add("TITLE");
queryColumns.add("PRIORITY");
queryColumns.add("STATE");
queryColumns.add("ENDDATE");
queryColumns.add("NUMBERATTRIBUTE1");
queryColumns.add("TEXTATTRIBUTE1");

45.4.5.5 Construct a List of OptionalInfo Items to be Returned from queryTasks()

This step is optional.

Per the API documentation, the OptionalInfo enumeration consists of additional, optional values that can be obtained with the task in the result set. These optional values include the available actions for a task, attachments, user comments, and so on. An example is shown in Example 45-20.

Example 45-20   Constructing a List of OptionalInfo Items

```
// List of optional info
// You can fetch any specified optionalInfo items from the Task object.
// For example: if you have specified "CustomActions", you can retrieve
// it using task.getSystemAttributes().getCustomActions();
// "Actions" (All Actions) - task.getSystemAttributes().getSystemActions()
// "GroupActions" (Only group Actions: Actions that can be permuted by the user
// as a member of a group).
// "ShortHistory" - task.getSystemAttributes().getShortHistory()
List optionalInfo = new ArrayList();
optionalInfo.add("Actions");
```

45.4.5.6 Invoke queryTasks() with the Attribute Lists

Now that the attribute classes have been constructed to constrain, order and specify the attributes returned in the query, invoke the queryTasks() method.

The query service API has the method signature shown in Example 45-21.

The queryTasks() method returns a list of tasks that match the predicate and ordering criterion.

Example 45-22 shows how to invoke queryTasks() using the previously constructed attribute lists.

More information on paging is available in the API documentation. (Look for the text "How to use paging.")

Example 45-21   Query Service API

```java
queryTasks(IWorkflowContext ctx, java.util.List displayColumns,
        java.util.List<ITaskQueryService.OptionalInfo> optionalInformation,
        ITaskQueryService.AssignmentFilter assignmentFilter,
        java.lang.String keywords,
        Predicate predicate,
        Ordering ordering,
        int startRow,
        int endRow)
```

Example 45-22   Invoking queryTasks() with the Previously Constructed Attribute Lists

```
List tasksList = querySvc.queryTasks(wfCtx,
        queryColumns,
        optionalInfo,
        ITaskQueryService.ASSIGNMENT_FILTER_MY_AND_GROUP,
        keyword,
```
**45.4.5.7 Iterate through the Result Set**

The method `queryTasks` returns a list of task objects. Use standard Java iteration to iterate through the list. Then, invoke various accessors to obtain the attributes specified in the query column list. Example 45-23 shows how to iterate through the result set.

**Example 45-23  Iterating through the Result Set**

```java
if (tasksList != null) { // There are tasks
    str = str + tasksList.size() + ":";
    Task task = null;
    for (int i = 0; i < tasksList.size(); i++) {
        task = (Task) tasksList.get(i);
        str = str + task.getSystemAttributes().getTaskNumber() + "/" +
        task.getSystemAttributes().getTaskId();
        System.out.println("Task Number: "+
        task.getSystemAttributes().getTaskNumber());
        System.out.println("Task Id: " + task.getTaskId());
        System.out.println("Title: "+ task.getTitle());
        System.out.println("Priority: " + task.getPriority());
        System.out.println("State: "+ task.getSystemAttributes().getState());
        // Retrive any Optional Info specified
        // Use task service, to perform operations on the task
        str = str + ":";
    }
}
```

**45.4.5.8 Programmatically Set the Task Outcome**

After you have obtained one or more task objects through the query service, approve or reject the tasks by calling the task service `updateTaskOutcome` method from the `ITaskService` API, as shown in the following examples:

- **Example 45-24**
- **Example 45-25**
- **Example 45-26**
- **Example 45-27**
- **Example 45-28**
- **Example 45-29**

**Example 45-24  Programmatically Setting the Task Outcome**

```java
updateTaskOutcome(IWorkflowContext context, Task task, java.lang.String outcome)
// Set the outcome of the task.
```

**Example 45-25  Setting the Outcome of the Task Reference on the Single Server Task Service**

```java
'APPROVE'.taskSvc.updateTaskOutcome(wfCtxt, t, "APPROVE");
```
Example 45-26  Setting the Outcome of the Task Reference on the Single Server Task Service

'REJECT'.taskSvc.updateTaskOutcome(wfCtxt, t, "REJECT");

Example 45-27  Obtaining a Single Task Reference Using the Non-Federated Query Service

getTaskDetailsById, then setting the outcome to 'APPROVE'.Task t =
querySvc.getTaskDetailsById(wfCtxt, "0a6d287a-9849-4e5e-914b-805706d6b9d9");
taskSvc.updateTaskOutcome(wfCtxt, t, "APPROVE");

Example 45-28  Obtaining a Single Task Reference Using the Non-Federated Query Service

getTaskDetailsByNumber, then setting the outcome to 'REJECT'.Task t =
querySvc.getTaskDetailsByNumber(wfCtxt, 200140 );
taskSvc.updateTaskOutcome(wfCtxt, t, "REJECT");

Example 45-29  Updating a Single Task Outcome on the Federated Query Service

Map<String, IWorklowContext> ctxMap = fedWFCtx.getWorkflowContextMap()
String serverName = task.getServerName();
IWorkflowContext taskWfCtx = ctxMap.get(serverName);
if (taskWfCtx !=null)

45.5 Other Approaches

Programmatically, it is possible to use a SOAP-based client interface and manually set
endpoint and credential information in the code from a product-specific table. This
approach is not recommended as the SOAP interface may negatively affect
performance liability. Furthermore, managing endpoint or credential information
yourself is not recommended. Endpoint implementation and credential provisioning
are best facilitated by centralized endpoint management and identity propagation.

45.6 Securing the Design Pattern

Secure your human workflow client service in the manner appropriate for the type of
code you are developing. For more information about securing your application, see
Securing Web Services Use Cases .

45.7 Verify the Deployment

Validating your implementation involves deploying human tasks to a SOA domain, in
the following steps.

- Deploying the Human Task
- Deploying Programmatic Task Functionality
- Invoking Programmatic Task Functionality

45.7.1 Deploy the Human Task

Upon deployment of the SOA composite containing the human task metadata, the
human workflow infrastructure registers the task by name and namespace. After the
task is deployed and registered, your code or BPEL can initiate the task and facilitate
task resolution through the code or worklist.
45.7.2 Deploy Programmatic Task Functionality
Deploy the application containing the task service client code. Make sure to deploy the
SOA composite with a task before deploying the human workflow task client service.
This ensures that the SOA composite has been deployed before creating a task
instance.

45.7.3 Invoke Programmatic Task Functionality
You can use Oracle ADF UI, an ADF Business Components service or an Oracle
Enterprise Scheduler Java job to implement the task client service functionality.
Submit or invoke the task functionality as you normally would, and use the Worklist
application to confirm that tasks have been created and updated. You can access the
Worklist application at the following URL:

45.7.4 Related Links
The following documents provide additional information related to subjects discussed
in this section:

- For more information about deploying human tasks, see "Part V: Using the
  Human Workflow Service Component" in the Developing SOA Applications with
  Oracle SOA Suite.

- For more information about deploying programmatic task functionality, see
  "Deploying and Managing SOA Composite Applications" in the Administering

45.8 Troubleshoot the Use Case
Following are some suggestions for troubleshooting task data and the Java code in the
use case.

45.8.1 Troubleshooting Task Data
In some cases, tasks may have been initiated but the attributes required for the task
have not been set. When this happens, the task may not display in the worklist.
Alternatively, it may be assigned to the wrong user, or to no user at all.
Check the WFTASK table for tasks such as these.

45.8.2 Troubleshooting Java Code
Use Oracle Fusion Middleware Extensions for Applications AppsLogger APIs to
write execution and exception details to the diagnostic logs. You can also use the
Oracle JDeveloper remote debugger to remotely connect to the runtime JVM and step
through your code.

45.9 What You May Need to Know About Implementing Email Notification
for an Oracle ADF Task Flow for a Human Task

- This approach uses the RMI interface to the human workflow task services for
  performance and identity propagation.
• Endpoint information for the RMI URL is stored in the Oracle Fusion Middleware Extensions for Applications taxonomy tables and available in JAXB object form by the taxonomy APIs.

• Use of paging (25 or 50) to limit result set size in task query or federated task query calls is necessary for performance reasons.
This chapter describes what to do if your SOA composite includes a human task, and you need to define an Oracle ADF task flow for a human workflow for users to interact with the task.

**When to implement:** If your SOA composite includes a human task, then you need to define an Oracle ADF task flow for a human workflow for users to interact with the task.

**Design Pattern Summary:** If your SOA composite includes a human task, then you need a way for users to interact with the task. The integrated development environment of Oracle SOA Suite includes Oracle Application Development Framework (Oracle ADF) for this purpose. With Oracle ADF, you can design a task display form that depicts the human task in the SOA composite.

**Involved components:**
- Oracle ADF task flow for human tasks
- Human task service component

### 46.1 Introduction to the Recommended Design Pattern

When invoking a human task from a SOA composite, an interface is required so that end users can interact with the task. You can use Oracle ADF to develop an interface that displays the human task within the SOA composite.

### 46.2 Other Approaches

There are no other supported approaches to this use case.

### 46.3 Example

The sample code for this use case can be downloaded from Oracle SOA Suite samples.

### 46.4 How to Implement an Oracle ADF Task Flow for a Human Task

This section describes the procedure used to invoke an Oracle ADF task flow for a human task. The procedure includes tasks that are detailed in the following sections:

Implementing an Oracle ADF task flow for a human task involves the following tasks:
- Creating an Oracle ADF task flow.
- Creating a user interface.
• Confirming the classpath, libraries and tag libraries.
• Implementing product-specific sections.
• Implementing a task detail with contextual area.
• Implementing email notification.
• Displaying localized translated data.
• Displaying rows in the approval task.
• Configuring a deployment profile.

Before You Begin:
Ensure that you do the following:

• Define your human workflow task definition in SOA workspace. The TASK file
definition and schemas are referenced when creating the Oracle ADF task flow for
human tasks. The TASK file defines the data controls used in your task detail
page.

• Define the UI and uiModel project following the directory structure, package
structure and naming standards. Although the task detail page is associated with
a human task definition in your SOA composite, add to source control the UI
project containing the Oracle ADF task flow for human task definition with the
Oracle ADF workspace, not the SOA workspace.

• If you plan to create other UI and uiModel projects unrelated to Oracle ADF task
flow for human tasks, you can create them in the same LBA directory as the UI
and uiModel projects associated with an Oracle ADF task flow for human tasks.
However, it is recommended to keep the other UI and uiModel projects in
separate projects, using the optional <Context> to differentiate them.

46.4.1 Creating an Oracle ADF Task Flow
The first step in the use case is to create an Oracle ADF task flow.

To create an Oracle ADF task flow:

1. Right-click your UI project and choose New.

2. In the New Gallery window, select Web Tier > JSF > ADF Task Flow Based on
Human Task and click OK.

3. In the SOA Resource Browser dialog, select the TASK file location and target TASK
file.

   This creates the data control definition.

4. Create a bounded task flow. From the Create Task Flow dialog, enter a name for
the task flow and click OK.

Note:
There is no established task flow file-naming standard.
This creates the task flow containing a View component with the default name taskDetails1_jspx, as shown in Figure 46-1. Rename the view activity to something meaningful to you.

Figure 46-1  Task Flow with View Component

5. Double-click the view activity in the task flow. From the Create JSF Page dialog that displays, modify the file name or directory location as needed, and click OK.

6. When placing a bounded task flow on a JSPX page, make sure to handle exceptions in the bounded task flow. If the exception is propagated to the unbounded task flow, the bounded task flow may exit, causing the JSPX page to behave unpredictably.

Use the template with the ID ExceptionHandlerTaskFlowTemplate in the JSPX page to avoid any unpredictable behavior. The template is located in the UIComponents-View.jar, as follows: /oracle/apps/fnd/applcore/patterns/uishell/templates/ExceptionHandlerTaskFlowTemplate.xml.

Note:

Use the Property Inspector to add the template to the page, as shown in Figure 46-2.
46.4.2 Create a User Interface for the Human Task

Use the drop handler template to create a user interface for the human task.

To create a user interface:

1. Create a task detail UI and navigate to Application Navigator > Data Controls > Task Data Control Name > getTaskDetails > Return > Task.

2. Drag and drop the task data control to your JSPX page and select Create > Human Task > Complete Task with Payload, as shown in Figure 46-3.
The Edit Action Binding (getTaskDetails) dialog is displayed.

3. In the Edit Action Binding (getTaskDetails) dialog, click OK.

The Edit Action Binding (UpdateActions) dialog displays, as shown in Figure 46-4.

4. In the Edit Action Binding (UpdateActions) dialog, click OK.

5. In the Source view of the JSPX page, verify that `<af:panelHeader>` displays as the top most component in the `<af:form>` component. Figure 46-5 shows the Source view of a sample JSPX page.
6. Verify that the Applications Core (ViewController) library is included in the UI project class path.

7. Verify that the following distributed libraries are included in the JSP Tag Libraries for the project:
   - Trinidad HTML Components,
   - Workflow Tags 1.0,
   - worklistComponents 1.0,
   - Applications Core (ViewController).

46.4.3 Implement Product-Specific Sections

Product-specific sections include the following:

- Instructions
- Details
- Recommended Actions
- <PLACE APPLICATION SPECIFIC CONTENT HERE>
- Related Links
- Comments and Attachments
- History
46.4.3.1 How to Add Instructions

You can add instructions just before the Details section.

To include instructions in the panelHeader:

Add the component `<af:outputText>` before the Details section, as shown in Example 46-1.

**Example 46-1 Adding instructions above the Details section**

```html
<af:panelGroupLayout layout="vertical" id="pgl3">
  <f:facet name="separator">
    <af:spacer width="15" height="15" id="s6"/>
  </f:facet>
  <af:outputText value="[Instruction text goes here]"/>
  <af:showDetailHeader size="1" text="#{resources.DETAILS}" shortDesc="#{resources.TASK_HEADER}" disclosed="true" id="sdh1"/>
</af:panelGroupLayout>
```

46.4.3.2 How to Modify Details

The Details section, as shown in Figure 46-6, contains the required human task information displayed in the Approvals, Request for Action, and FYI patterns in the first column and displays optional product-specific header information for the task.

**Figure 46-6 Details Section**

![Details Section](image)

Do not modify the code in the first column, which corresponds to the first `<trh:cellFormat>` with id="cf1", as shown in Figure 46-6.

Do any or all of the following:

- If required, change the text attribute for the `<af:showDetailHeader>` component. The default is set to "#{resources.DETAILS}" for example, Details.
- If your page does not display product-specific information in this section, skip this section and continue to the next section.
• If your page displays product-specific information, add it to the third
  <trh:cellFormat> component, which corresponds to the
  <trh:cellFormat> with id="cf6" shown in Example 46-2.

  **Note:**
  Based on the UX specification, the task information displayed in the first
column should only be displayed in the first column and should not wrap into
the third column.

• If your page requires an additional column of product-specific information, take
the following steps.
  Add a column for spacing, <trh:cellFormat id="cf7">. Add a column to
display the product-specific information, <trh:cellFormat id="cf8">.
  Change the width to 33%, for <trh:cellFormat> components with id="cf1",
id="cf6", and id="cf8" as shown in Example 46-3.

**Example 46-2  Modify the third <trh:cellFormat> component**

```xml
<af:showDetailHeader size="1" text="#{resources.DETAILS}" shortDesc="#{resources.TASK_HEADER}" disclosed="true" id="sdh1">
  <f:facet name="toolbar"/>
  <trh:tableLayout width="98%" id="tl1">
    <trh:rowLayout id="rl2">
      <trh:cellFormat width="50%" valign="top" id="cf1">
        ... First column of task-specific fields, do not modify ...
      </trh:cellFormat>
      <trh:cellFormat id="cf5">
        <af:spacer width="15" height="15" id="s3"/>
      </trh:cellFormat>
      <trh:cellFormat width="50%" valign="top" id="cf6">
        ... Second column containing product-specific fields ...
      </trh:cellFormat>
    </trh:rowLayout>
  </trh:tableLayout>
</af:showDetailHeader>
```

**Example 46-3  Adding product-specific information**

```xml
<af:showDetailHeader size="1" text="#{resources.DETAILS}" shortDesc="#{resources.TASK_HEADER}" disclosed="true" id="sdh1">
  <f:facet name="toolbar"/>
  <trh:tableLayout width="98%" id="tl1">
    <trh:rowLayout id="rl2">
      <trh:cellFormat width="33%" valign="top" id="cf1">
        ... First column of task-specific fields, do not modify ...
      </trh:cellFormat>
      <trh:cellFormat id="cf5">
        <af:spacer width="15" height="15" id="s3"/>
      </trh:cellFormat>
      <trh:cellFormat width="33%" valign="top" id="cf6">
        ... Second column containing product-specific fields ...
      </trh:cellFormat>
      <trh:cellFormat id="cf7">
        <af:spacer width="15" height="15" id="s3"/>
      </trh:cellFormat>
      <trh:cellFormat width="33%" valign="top" id="cf8">
```
46.4.3.3 How to Modify Recommended Actions

The Recommended Actions section, as shown in Figure 46-7, is used in the Information Only pattern.

Figure 46-7  Recommended Actions Section

The Oracle ADF code for the Recommended Actions <af:showDetailHeader> component is shown in Example 46-4.

- If your page does not display the Recommended Actions section, remove the <af:showDetailHeader> component with text="#{resources.RECOMMENDED_ACTIONS}" and continue to the next section.
- If your page does display the Recommended Actions section, add the actions and appropriate links to this section.
Note:
Translation and accessibility standards state that individual words should not be implemented as links.

**Example 46-4  Code for Recommended Actions**

```xml
<af:showDetailHeader size="1" id="recommendedActionsHeader"
    text="#{resources.RECOMMENDED_ACTIONS}" disclosed="true">
    <f:facet name="info"/>
    <f:facet name="legend"/>
    <f:facet name="menuBar"/>
    <f:facet name="toolbar"/>
    <f:facet name="context"/>
</af:showDetailHeader>
```

### 46.4.3.4 How to Modify `<PLACE APPLICATION SPECIFIC CONTENT HERE>`

The `<PLACE APPLICATION SPECIFIC CONTENT HERE>` section is a placeholder for product-specific information such as the Purchasing Line section in the Approval Page Details pattern.

The Oracle ADF code for the `<PLACE APPLICATION SPECIFIC CONTENT HERE>` component is shown in **Example 46-5**.

Do any or all of the following:

- If your page does not have an application-specific section, then remove the `<af:showDetailHeader>` component with `text="<PLACE APPLICATION SPECIFIC CONTENT HERE>"` and continue to the next section.

- Modify the text attribute of the `<af:showDetailHeader>` component.

- Modify the body of the `<af:showDetailHeader>` component to meet your requirements.

Note:
The **Complete Task with Payload** drop handler adds an `<af:inputText>` component for each of the task payload fields by default. Remove these fields if they are not required.

- Add links to the bottom of this section, as shown in the pattern. The link implementation instructions are discussed in **How to Implement Links**.

- If additional product-specific sections are required, add them directly below the `<af:showDetailHeader>` section. Ensure that the size of the additional `<af:showDetailHeader>` components is set to 1.

**Example 46-5  Oracle ADF code for the application specific content section**

```xml
<af:showDetailHeader size="1" id="applicationContentHeader" text="<PLACE APPLICATION SPECIFIC CONTENT HERE>" disclosed="true">
    <af:panelGroupLayout id="payload_panel" layout="vertical"
        shortDesc="#{resources.CONTENTS}"
        />
    <af:panelFormLayout id="pfl1">
        <af:inputText value="#{bindings.PayloadInput1.inputValue}"
            label="#{bindings.PayloadInput1.hints.label}"/>
    </af:panelFormLayout>
</af:showDetailHeader>
```
46.4.3.5 How to Implement Links

Use the `af:goLink` component and specify `targetFrame="_blank"` to implement the related link.

To construct the target URL, use the API `oracle.apps.fnd.applcore.patterns.uishell.context.UIShellContext.getURL`. This API can still be used even though the task detail page does not implement the UI Shell. Enter a non-null `webApp` parameter to generate the full URL including the host name and port.

The resulting code should be similar to that displayed in Example 46-6.

**Example 46-6 Code that Results from Implementing Links**

```java
UIShellContext.getURL(java.lang.String viewId,
java.lang.String webApp,
java.lang.String pageParametersList,
java.lang.String navTaskFlowId,
java.lang.String navTaskKeyList,
java.lang.String navTaskParametersList,
java.lang.String navTaskLabel,
FndMethodParameters methodParameters)
```

46.4.3.6 How to Modify Comments and Attachments

The Comments and Attachment sections are used to store comments and attachments associated with the Approval and Request for Action patterns.

Do any of the following:

- If your page requires both comments and attachments, then leave it as is and continue to the next section.

- If your page does not require any comments or attachments, as in the FYI pattern, remove the `<af:switcher>` component with `facetName="#{pageFlowScope.bpmClientType}"` and its facets and continue to the next section.

- If your page requires the comments section only, move the `<af:showDetailHeader>` component with `text=#{resources.COMMENTS}` from the switcher facet with name="notificationClient" so that it is a peer of the switcher with `facetName="#{pageFlowScope.bpmClientType}"`. Then delete the `<af:switcher>` component with `facetName="#{pageFlowScope.bpmClientType}"` and its facets. The resulting code is shown in Example 46-7.

- If you require additional instructions in the Attachments dialog, then add an `<af:outputText>` component after `<af:outputText>` with value="#{resources.UPLOAD_FILE_CAVEAT}"", as shown in Example 46-8.
Example 46-7  Modifying comments and attachments

```xml
<af:showDetailHeader size="1" id="relatedLinksHeader"
  text="#{resources.RELATED_LINKS}"
  disclosed="true">
  ...
</af:showDetailHeader>

<af:showDetailHeader size="1" id="showDetailHeader1"
  text="#{resources.COMMENTS}"
  disclosed="true">
  ...
</af:showDetailHeader>

<af:showDetailHeader size="1" id="historyHeader"
  text="#{resources.HISTORY}"
  disclosed="false">
  ...
</af:showDetailHeader>
```

Example 46-8  Adding additional instructions to the Attachments dialog

```xml
<af:popup id="popupAddAttachmentDialog">
  <af:dialog title="#{resources.ADD_ATTACHMENT}" okVisible="false"
    cancelVisible="false" closeIconVisible="false" id="d2">
    ...
  </af:dialog>
</af:popup>
```

46.4.3.7 How to Modify Related Links

The Related Links section is used in the Information Only pattern.

Example 46-9 shows the Oracle ADF code for the Related Links component.

- If your page does not display the Related Links section, then remove the `af:showDetailHeader` component with text="#{resources.RELATED_LINKS}" and continue to the next section.

- If your page displays the Related Links section, add the appropriate links to this section. The link implementation instructions are described in How to Implement Links.

Example 46-9  Modifying Related Links

```xml
<af:showDetailHeader size="1" id="relatedLinksHeader"
  text="#{resources.RELATED_LINKS}" disclosed="true">
  <f:facet name="info"/>
  <f:facet name="legend"/>
  <f:facet name="menuBar"/>
  <f:facet name="toolbar"/>
```
46.4.3.8 How to Modify History

The History section displays the tabular and graphical displays of the task history. Do any of the following:

- If your page requires the history section, then leave it as is.
- If your page does not require the history section, remove the `<af:showDetailHeader>` component and its facets and child components, as shown in Example 46-10.

**Example 46-10  Modifying task history**

```xml
<af:showDetailHeader size="1" id="historyHeader" text="#{resources.HISTORY}" disclosed="false">
  <f:facet name="info"/>
  <f:facet name="legend"/>
  <f:facet name="menuBar"/>
  <f:facet name="toolbar"/>
  <f:facet name="context"/>
  <af:panelGroupLayout layout="vertical" id="pgl7">
    <wlc:taskHistory initParam="#{aleComponentBean.comp}" showTabularView="true" showGraphicalView="true" id="th1"/>
  </af:panelGroupLayout>
</af:showDetailHeader>
```

46.4.4 Implementing a Task Detail with Contextual Area

If you are implementing a task detail page with a contextual area, do not use the UIShellMainArea template when creating the JSF page. Instead, create the JSF page without a template.

Use `<trh:tableLayout>`, `<trh:rowLayout>` and `<trh:cellFormat>` to configure the layout of the local and contextual areas. For the local area, follow the steps outlined in the following sections:

- Creating an Task Flow
- Creating a User Interface for the Human Task
- Implementing Product-Specific Sections
- Implementing a Task Detail with Contextual Area

46.4.5 Implement Email Notification

The email version of the task details are often called notifications, task notifications or email notifications. Email notifications are viewed in email clients.

The goal in implementing the email version of the task detail page is to use the same Oracle ADF task flow for human tasks definition for both the online and email versions. As both versions are nearly identical, you can implement both as a single page to avoid dual maintenance.
46.4.5.1 Before You Begin

Ensure that the TASK file is configured so that email notifications are actionable and task attachments are added to the email as email attachment, as shown in Figure 46-8.

Figure 46-8 Notification Settings

<table>
<thead>
<tr>
<th>Task Status</th>
<th>Recipient</th>
<th>Notification Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assign</td>
<td>Assignees</td>
<td></td>
</tr>
<tr>
<td>Complete</td>
<td>Initiator</td>
<td></td>
</tr>
<tr>
<td>Error</td>
<td>Owner</td>
<td></td>
</tr>
</tbody>
</table>

46.4.5.2 Determine the Implementation Approach

Select an implementation approach from the following:

- If the JSPX page that you defined for your online version contains only the following supported components, then you can use your existing JSPX page for both online and email versions.
  - af:column
  - af:commandLink
  - af:document
  - af:goLink
  - af:image
  - af:inputText
  - af:inputComboBoxListOfValues
  - af:inputDate
  - af:inputListOfValues
  - af:inputNumberSlider
  - af:inputNumberSpinbox
  - af:inputRangeSlider
  - af:outputText
  - af:panelHeader
  - af:panelLabelAndMessage
• If the online version of your JSPX page includes many interactions to be made available in the email notification, then you will need to build a second JSPX page for your email notification.

• Use an If component in the JSPX page to ensure that only supported components are rendered in the email version. For more information, see Using an If Component.

46.4.5.3 Using an If Component

Introduce an If component in your JSPX page when you have components that are not supported in the email notification.

To use an If component in your JSPX page:

1. Move the unsupported components in the <f:facet> with name="online".

2. Add alternative rendering logic using the supported components in <f:facet> where name="notificationClient", as shown in Example 46-11.

3. If it is not a notificationClient, the value of bpmClientType is null. Continue to Fine-Tuning the Emailable Page.

Example 46-11  Using an If component

<af:if defaultFacet="online"
   facetName="#{pageFlowScope.bpmClientType}"
>
   <f:facet name="online">
     ... Place code rendered in online version here ...
   </f:facet>
   <f:facet name="notificationClient">
     ... Place code rendered in email version here ...
   </f:facet>
</af:if>

46.4.5.4 Using a Separate View for Online and Email Versions

You can enable separate views for online and email versions for notifications.
To enable separate views for online and email notifications:

1. In the task flow, add a View Activity for the email version.
   a. Add a new view to the task flow definition and rename it appropriately.
   b. Define a control flow from the newly introduced view to the existing task flow return activity called taskReturn.
   c. Modify the transition value to closeTaskFlow.

2. Add a router.
   a. Add a new router activity and rename it appropriately.
   b. In the property inspector, set the default Outcome attribute to online.
   c. Add two cases, as shown in Example 46-12.
   d. Define a control flow case from the newly introduced router to the view associated with the online version. For the transition value, specify online.
   e. Define a control flow case from the newly introduced router to the view associated with the email version. For the transition value, specify email.

3. Set the newly introduced router as the default activity, as shown in Figure 46-9.

**Figure 46-9 Setting Default Activity**

![Diagram of task flow with router and views]

**Example 46-12 Adding two cases to the switcher**

Expression = "#{pageFlowScope.bpmClientType=="notificationClient"}" and Outcome = "email"

Expression = "#{empty pageFlowScope.bpmClientType}" and Outcome = "online"
46.4.5.5 Fine-Tuning the Emailable Page

To correctly render your JSPX page in email mode, you may need to take any or all of following steps to fine-tune the email page.

- Use an `<af:outputText>` component to render the instruction text, as there is no style class for instruction text based on the `helpTopicId` of `<af:panelHeader>` or `<af:showDetailHeader>`.

- In the `<af:showDetailHeader>` component, if the disclose property is set to false, the disclosure is closed in the page to be emailed. Add an EL expression to the `disclosed` property so that the property is true when in email mode, as shown in Example 46-13.

- Test your page to determine whether the width of `<af:table>` is acceptable. If the table is not wide enough, set the `<af:column>` width attribute.

Example 46-13 Adding an EL expression to the disclosed property

```jsp
<af:showDetailHeader size="1"
    id="histHd"
    text="#{resources.HISTORY}"
    disclosed="#{pageFlowScope.bpmClientType == 'notificationClient'}"
...
```

46.4.6 Displaying Localized Translated Data

If you are adding product-specific code to the template, ensure that your product-specific code adheres to the localization standards for Oracle ADF user interfaces.

If you are displaying a task payload attribute in your JSPX page, you must override the EL expression for the label attribute for each of the payload attributes. For example, in the automatically generated Oracle ADF code shown in Example 46-14, replace the "#{bindings.PayloadInput1.hints.label}" so that it refers to translated text in the Xliff resource bundle associated with your UI project.

Note:

Two strings in the Oracle ADF code template that store their translations in the resource bundle associated with the TASK file (as opposed to the UI project). Do not define the translations in the resource bundle associated with your UI.

- **Title**: Define the title in the task definition using the `Translation` setting. Ensure a translation is provided in a Java resource bundle. Do not use the properties file to store translated text.

- **Custom actions**: Ensure a translation is provided in a Java resource bundle. Do not use the properties file to store translated text.

For more information about translation resource bundles, see the section entitled "How to set up Resource Bundles for Translation of Your Customizations" in Creating Customizable Applications.
46.4.7 Displaying Rows in the Approval Task

If you want to display the rows in the business object that are included in the approval task, use the collection target data in the task data control to determine the rows to be included. Render that information within your product-specific regions.

To use a collection target:

- In Oracle JDeveloper, select Application Navigator > Data Controls > Task Flow Name > getTaskDetails(String, String, String) > Return > Task > System Attributes > Collection Target.

Regardless of whether aggregation is enabled, the collection target contains information on the rows that are being approved or rejected by the approver.

46.4.8 Configure a Deployment Profile

Create a deployment profile as you would for any other Oracle ADF UI project. For more information about configuring a deployment profile, see the section entitled "How to Create Deployment Profiles for Standalone WebLogic Server Deployment" in Setting Up Your JDeveloper Application Workspace and Projects.

To configure a deployment profile:

1. Generate an Oracle ADF library for your UI project containing the Oracle ADF task flow for a human task definition.

2. Add the Oracle ADF library to the SuperWeb project.
   Open the web.xml file for the SuperWeb project and add the code shown in Example 46-15.

3. Under the default source location, create the hwtaskflow.xml file and merge all the notification task flow details. For example, if you are adding a new ADF task flow for a human task to a project defined in a workspace called fusionapps/fin/components/Payables.jws, then you would take the following steps.

   a. In the SuperWeb project of your workspace, manually edit the hwtaskflow.xml file. In this example, the hwtaskflow.xml file is located in the following directory: fusionapps/fin/components/payables/PayablesSuperWeb/afmsrc/hwtaskflow.xml.
b. Under the `<hwTaskFlows>` element, add the relevant `<hwTaskFlow>` elements, as shown in **Example 46-16**.

**Example 46-15   Adding the Oracle ADF library to the SuperWeb project**

```xml
<filter>
  <filter-name>WorkflowFilter</filter-name>
  <filter-class>
    oracle.bpel.services.workflow.client.worklist.util.WorkflowFilter
  </filter-class>
</filter>

<filter-mapping>
  <filter-name>WorkflowFilter</filter-name>
  <url-pattern>/faces/*</url-pattern>
</filter-mapping>

<servlet>
  <servlet-name>IntegrateTaskFlowWithTask</servlet-name>
  <servlet-class>
    oracle.bpel.services.workflow.client.worklist.servlet.IntegrateTaskFlowWithTask
  </servlet-class>
  <load-on-startup>2</load-on-startup>
</servlet>

<servlet>
  <servlet-name>secureNotificationServlet</servlet-name>
  <servlet-class>
    oracle.bpel.services.workflow.client.worklist.servlet.SecureNotificationServlet
  </servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>IntegrateTaskFlowWithTask</servlet-name>
  <url-pattern>/integratetaskflowwithtask</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>secureNotificationServlet</servlet-name>
  <url-pattern>/notification/secure</url-pattern>
</servlet-mapping>

<context-param>
  <param-name>oracle.adf.view.rich.security.FRAME_BUSTING</param-name>
  <param-value>differentDomain</param-value>
</context-param>

**Example 46-16   Add the `<hwTaskFlow>` elements**

```xml
<xml version='1.0' encoding='UTF-8'>
<hwTaskFlows
  xmlns="http://xmlns.oracle.com/bpel/workflow/hwTaskFlowProperties">
  <ApplicationName>worklist</ApplicationName>
  <LookupType>LOCAL</LookupType>
  <TaskFlowDeploy>false</TaskFlowDeploy>
  <SoaServer>
    <ejbProviderUrl/>
    <aliasKeyName/>
    <keyName/>
    <connectionName/>
  </SoaServer>
</hwTaskFlows>
```
46.5 Securing the Design Pattern

To secure this pattern, follow the instructions described in Securing Web Services Use Cases.

You may want to implement the following security tasks:

- Secure task flows (bounded)
- Secure page fragments
- Secure actions

46.6 Verifying the Deployment

Verifying the deployment involves defining JNDI and foreign JNDI for the non-SOA Oracle WebLogic Server, as well as defining a grant for bpm-services.jar.
In this use case, the Oracle ADF task flow for human tasks is deployed to a non-SOA server as described here, including configuring foreign JNDI providers. Alternatively, you can define a connection to the Oracle SOA Suite server using a deployment script.

You can deploy the Oracle ADF task flow for a human task to a SOA server.

**To verify the deployment:**

1. Deploy the `oracle.soa.workflow` shared library to the non-SOA Oracle WebLogic Server.
   - Navigate to `http://remote_hostname:remote_port/console`, where `remote_hostname` and `remote_port` are the host name and port for the remote non-SOA WebLogic server.
   - Select **Deployments** and click **Install**.
   - In the **Path** field, make sure the following value is specified:
     
     ```
     <jdev_install>/fmwtools/fmwtools_home/jdeveloper/soa/modules/oracle.soa.workflow_11.1.1
     ```
   - Select `oracle.soa.workflow.jar` and click **Finish**.
   - Confirm that the `oracle.soa.workflow(11.1.1,11.1.1)` library is active.

2. Define the foreign JNDI on the non-SOA Oracle WebLogic Server.
   - Navigate to `http://remote_hostname:remote_port/console`, where `remote_hostname` and `remote_port` are the host name and port for the remote non-SOA WebLogic server.
   - Navigate to **Domain Structure > Services > Foreign JNDI Providers** and click **New**.
   - Enter the name `ForeignJNDIProvider-SOA`, and click **OK**.
   - Click **ForeignJNDIProvider-SOA**.
   - Press **Enter** and then click **Save**.
   - Fill in the following for the SOA Oracle WebLogic Server.
     
     **Initial Context Factory**: `weblogic.jndi.WLInitialContextFactory`
     
     **Provider URL**: Enter the URL of the `soa-infra` application, using the following format: `t3://SOA_hostname:SOA_port/soa-infra`
     
     **User/Password**: Enter an administrator username and password for the server.

     **Note:**
     
     The provider URL refers to the `soa-infra` application, not the domain. Do not change `soa-infra`.

3. Define the JNDI links on the non-SOA Oracle WebLogic Server.
a. Navigate to http://remote_hostname:remote_port/console, where remote_hostname and remote_port are the host name and port for the remote non-SOA Oracle WebLogic Server.

b. Navigate to Domain Structure > Services > Foreign JNDI Providers and click ForeignJNDIProvider-SOA.

c. Select the Link tab and click New.

d. Press Enter and click OK. Enter the following values:
   - Name: RuntimeConfigService
   - Local JNDI Name: RuntimeConfigService
   - Remote JNDI Name: RuntimeConfigService
   - Specify ejb/bpel/services/workflow/ for ejb/bpel/services/workflow/TaskServiceBean and ejb/bpel/services/workflow/TaskMetadataServiceBean only.

e. Repeat steps 3.c and 3.d for the following JNDI values:
   - Name/Local JNDI Name/Remote JNDI Name: ejb/bpel/services/workflow/TaskServiceBean
   - Name/Local JNDI Name/Remote JNDI Name: ejb/bpel/services/workflow/TaskMetadataServiceBean
   - Name/Local JNDI Name/Remote JNDI Name: TaskReportServiceBean
   - Name/Local JNDI Name/Remote JNDI Name: TaskEvidenceServiceBean
   - Name/Local JNDI Name/Remote JNDI Name: TaskQueryService
   - Name/Local JNDI Name/Remote JNDI Name: UserMetadataService

4. On the remote non-SOA Oracle WebLogic Server, change jazn-data.xml (not system-jazn-data.xml) to include the grant for bpm-services.jar, as shown in Example 46-17.

5. Restart the remote non-SOA Oracle WebLogic Server.

6. Deploy your application containing the human task detail UI to the remote non-SOA Oracle WebLogic Server.

---

**Tips:**

When accessing the task in the Oracle Business Process Management Worklist, you may get the following message: “Details not available for this task.”

If so, take the following steps:

- Check the WFTASKDISPLAY table in the SOAINFRA schema for entries corresponding to the tasks. The entries should have the host and port number of the non-SOA Oracle WebLogic Server where the task detail page definitions are deployed.

- If there are no entries, check the log files for errors when deploying your task detail pages. Make sure that the grant for the bpm-services.jar is correctly defined.
Related Links
The following documents provide additional information related to subjects discussed in this section:

- For more information about using a deployment script to define a connection to the SOA server, see the section describing the workflow client configuration file (wf_client_config.xml) in the chapter "Introduction to Human Workflow Services" in the Developing SOA Applications with Oracle SOA Suite.

- For more information about deploying the Oracle ADF task flow for human task to a SOA server, see the chapter "Deploying SOA Composite Applications" in Developing SOA Applications with Oracle SOA Suite.

Example 46-17 Grant for bpm-services.jar

```xml
<grant>
  <grantee>
    <codesource>
      <url>file:${oracle.home}/soa/modules/oracle.soa.workflow_11.1.1/-</url>
    </codesource>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.security.jps.JpsPermission</class>
      <name>VerificationService.createInternalWorkflowContext</name>
    </permission>
    <permission>
      <class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
      <name>context=SYSTEM,mapName=BPM-CRYPTO,keyName=BPM-CRYPTO</name>
      <actions>read,write</actions>
    </permission>
    <permission>
      <class>oracle.security.jps.JpsPermission</class>
      <name>IdentityAssertion</name>
      <actions>*</actions>
    </permission>
  </permissions>
</grant>
```

46.7 Troubleshooting the Use Case

Following are some steps you can take to fix known issues.

- Specify oracle.soa.workflow.wc in weblogic-application.xml.
- Set the FRAME_BUSTING attribute in web.xml.
- Migrate from an earlier version of the drop handler template.
- Override the EL for the create button.

46.7.1 Specify oracle.soa.workflow.wc in weblogic-application.xml

By default, the drop handler generates a reference to the oracle.soa.workflow shared library in weblogic-application.xml. Override the entry in the weblogic-application.xml file so that it references oracle.soa.workflow.wc instead.
Example 46-18 shows a snippet of a sample `weblogic-application.xml` file with the correct reference to `oracle.soa.workflow.wc`.

**Example 46-18  Reference oracle.soa.workflow.wc**

```xml
<?xml version = '1.0' encoding = 'US-ASCII'?>
xmlns="http://www.bea.com/ns/weblogic/weblogic-application">
...
<library-ref>
    <library-name>oracle.soa.workflow.wc</library-name>
</library-ref>
...
</weblogic-application>
```

**46.7.2 Set the FRAME_BUSTING Attribute in web.xml**

If you receive a warning message about frame content not loading, modify `web.xml` to include the code shown in Example 46-19.

**Example 46-19  Set the FRAME_BUSTING attribute in web.xml**

```xml
<context-param>
    <param-name>oracle.adf.view.rich.security.FRAME_BUSTING</param-name>
    <param-value>differentDomain</param-value>
</context-param>
```

If your SOA runtime and task detail UI are running on different domains during development, set the value to `never` for testing purposes only.

**46.7.3 Migrate from an Earlier Version of the Drop Handler Template**

If your page is created using an earlier version of the drop handler template (which used `fnd:applicationPanel` and `af:panelStretchLayout`), take the following steps to migrate `fnd:applicationsPanel` to `af:panelHeader`.

**To migrate from an earlier version of the drop handler template:**

1. Open your JSPX page and, in the Structure pane, locate `af:panelStretchLayout` under `af:form`.

2. When you have found `af:panelStretchLayout`, add `af:panelHeader` above `af:panelStretchLayout` and set the text attribute to `text="#{binding.title.inputValue}"`.

3. In `fnd:applicationsPanel`, find the `actionButtonBar` facet and select the `af:toolbar` below it.
   Move `af:toolbar` under the toolbar facet of `af:panelHeader` added in the previous step.

4. Under the content facet of `fnd:applicationsPanel`, find `af:panelGroupLayout` and move it under `af:panelHeader`.

5. Find `af:panelStretchLayout` and delete it.
6. Save and test your work.

46.7.4 Override the EL for the Create Button

Override shortDesc EL for the Create button in the Comments section:

To override the EL for the Create button:

1. In the JSPX page, locate `<af:commandImageLink>` and find the action property value `#{popupBean.showCommentDialog}`.

2. For `<af:table>`, overwrite the shortDesc property value as follows:

   `shortDesc="#{resources.CREATE}"`.

3. Repeat step 2 for the second `<af:commandImageLink>` with the action property value `#{popupBean.showCommentDialog}`. Example 46-20 shows a sample illustrating `<af:commandImageLink>` after modifying the action property value.

Example 46-20  Modifying the action property value for `<af:commandImageLink>`

```xml
<af:commandImageLink id="adCmtBt" partialSubmit="true"
    actionListener="#{bindings.CreateInsert.execute}"
    action="#{popupBean.showCommentDialog}"
    shortDesc="#{resources.CREATE}"
    hoverIcon="/hw_images/new_ovr.png"
    visible="#{actionAvailable.isCommentUpdatable}"
    icon="/hw_images/new_ena.png"
    disabledIcon="/hw_images/new_dis.png"
    depressedIcon="/hw_images/new_dwn.png"/>
```
This chapter describes what to do when you require your SOA composite to subscribe to a business event published on another Oracle SOA Suite cluster.

**When to implement:** When you require your SOA composite to subscribe to a business event published on another Oracle SOA Suite cluster.

**Design Pattern Summary:** The Event Delivery Network (EDN) infrastructure supports business event publishing and subscription within a single Oracle SOA Suite cluster. To propagate a business event from one Oracle SOA Suite cluster to another, you must build a mediator-to-mediator bridge. The bridge queues the event message in a global queue from the Oracle SOA Suite cluster. The Oracle SOA Suite cluster publishes the event and then dequeues the event message from the Oracle SOA Suite cluster that subscribes to the event.

**Involved components:**
- Oracle Mediator
- Oracle Aqueue Adapter

### 47.1 Introduction to the Recommended Design Pattern

You can use business event publishing and subscription for asynchronous, loosely-coupled integration between two components such as an Oracle ADF UI application and a SOA composite. In most cases, the two components to be integrated are within the same family, and the same Oracle SOA Suite cluster is used to both raise and subscribe to the event. However, in some cases component integration goes across domains. This pattern is used to implement business event propagation from one Oracle SOA Suite cluster to another.

### 47.2 Potential Approaches

There are two other possible approaches to this use case.

- A global EDN queue can be shared across all Oracle Fusion Applications families. Although it supports cross-event subscription, the shared EDN feature is not recommended due to the durability of subscriptions. If any of the Oracle SOA Suite clusters sharing an event queue goes down, then the events will remain in the queue until the cluster goes back up and dequeues the event. If the cluster is offline for a long period, the queue could become quite large. In addition, a large number of events may need to be dequeued by each of the Oracle SOA Suite clusters even if there may no subscribers. This may affect performance.
WARNING:
This approach is not recommended.

- The composite with the event subscription can be deployed to the Oracle SOA Suite cluster which publishes the event.

### 47.3 Example

**CRM Subscribes to Event Published by HCM**

HcmUsersSpmlComposite deployed in the HCM SOA cluster publishes a post-processing event to report the success or failure of the process of creating a new user or assigning roles to a user. In order for CRM to perform additional actions based on the user request, the post-processing event must be propagated from HCM to CRM.

Table 47-1 lists the HCM composite components and values.

<table>
<thead>
<tr>
<th>Composite Components</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publishing Product</td>
<td>Per</td>
</tr>
<tr>
<td>Subscribing Product</td>
<td>Hz</td>
</tr>
<tr>
<td>Owning Product</td>
<td>Hz</td>
</tr>
<tr>
<td>XFAMILYPub composite name</td>
<td>FoundationPartiesPerToHzXFAMILYPubComposite</td>
</tr>
<tr>
<td>XFAMILYSub composite name</td>
<td>FoundationPartiesHzXFAMILYSubComposite</td>
</tr>
<tr>
<td>Recipient/Consumer</td>
<td>FoundationPartiesPerToHz</td>
</tr>
<tr>
<td>SOA workspace</td>
<td>CrmCommonSoa.jws</td>
</tr>
</tbody>
</table>

**Financials Subscribes to CRM Event**

When the ImportPartyData event is raised in the customer or supplier import program in TCA, the Financials PartyImportForTCALocService service must be invoked. The service creates a tax profile for a party which must be involved in financial transactions.

**HCM Subscribes to Event Published by CRM**

The resource directory feature in CRM raises the ActiveLdapRequestEvent event to make an LDAP request for provisioning the abstract role and security roles for newly provisioned user accounts to existing resources. Without processing these requests, resources will not be able to login to the resource directory used to maintain resource information such as phone numbers, email addresses, mail addresses, and so on. The HcmUsersSpmlComposite deployed to the HCM Oracle SOA Suite cluster subscribes to the event and processes the request.

Table 47-2 lists the CRM composite components and values.
### Table 47-2  CRM Composite Components and Values

<table>
<thead>
<tr>
<th>Composite Components</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publishing Product</td>
<td>Hz</td>
</tr>
<tr>
<td>Subscribing Product</td>
<td>Per</td>
</tr>
<tr>
<td>Owning Product</td>
<td>Hz</td>
</tr>
<tr>
<td>XFamilyPub composite name</td>
<td>FoundationPartiesHzToPerXFamilyPubComposite</td>
</tr>
<tr>
<td>XFamilySub composite name</td>
<td>FoundationPartiesPerXFamilySubComposite</td>
</tr>
<tr>
<td>Recipient/Consumer</td>
<td>FoundationPartiesHzToPer</td>
</tr>
<tr>
<td>SOA workspace</td>
<td>CrmCommonSoa.jws</td>
</tr>
</tbody>
</table>

The sample code for this use case can be downloaded from Oracle SOA Suite samples.

#### 47.4 How to Subscribe to a Cross-Family Business Event

To support cross-family event subscriptions, a global aqueue resides in the Oracle Fusion Applications schema to enqueue messages from one Oracle SOA Suite cluster and dequeue to another Oracle SOA Suite cluster. The messages contain information about the event as well as the publisher’s identity and application context information.

In addition to the aqueue, this pattern includes an `XFamilyPub` composite deployed to the Oracle SOA Suite cluster which publishes the cross-family event, and an `XFamilySub` composite deployed to the Oracle SOA Suite cluster which subscribes to the cross-family event. The `XFamilyPub` composite contains a mediator and an Aqueue adapter configured for the enqueue operation. The mediator subscribes to the event, transforms the event information and payload and enqueues a message to the Aqueue with the recipient set to the family subscribing to the event. The `XFamilySub` composite contains an Aqueue adapter configured for the dequeue operation with itself specified as the recipient. The mediator dequeues the message, filters the message based on the `Namespace` and `LocalName` elements in the message, transforms the payload and raises the event locally.

#### 47.4.1 Before You Begin

Your installation of Oracle JDeveloper must include the following database objects, Oracle WebLogic Server configurations and so on:

- Make sure that the database object `ACR_XFAMILY_EVENT_Q` is defined in the FUSION schema. Verify that the database object exists by navigating to the Database Navigator in Oracle JDeveloper, defining a connection to the FUSION schema and double-clicking Queues.

- In Oracle WebLogic Server, define the data source JDBC/ApplicationDBXA. Verify this configuration by running the Oracle WebLogic Server Console and navigating to Data Sources. (See bug 9712616.)

- In Oracle WebLogic Server, define a JDBC/XFamilyEventAqueue AqAdapter connection pool. Verify this configuration by running the Oracle WebLogic Server console and navigating to Domain Configurations > Your Deployment.
47.4.2 Determine the Composites to Be Defined

Following are the components to be defined for the pattern:

- **Publishing product**: The publisher product is the product that publishes the event.
- **Subscribing product**: The subscriber product is the product that subscribes to the event.
- **Owning product**: The owning product is the product that owns the `XFamilyPub` and `XFamilySub` composites.

The product team requiring the cross-family subscription owns both the `XFamilySub` and `XFamilyPub` composites. This pattern assumes the composites are defined at the product level. It is also possible to define the `XFamilySub` and `XFamilyPub` at the family level instead.

Determine whether an `XFamilySub` or `XFamilyPub` composite has been defined for your product. If they have not been defined, then follow these guidelines for naming the `XFamilySub` and `XFamilyPub` composites. The naming guidelines help make cross-family event composites easily identifiable and self-descriptive.

These composites can reside in any LBA in your product.

### Composite Naming Conventions

The composite names must adhere to the following guidelines:

1. **Start with the LBA short name.** You can define the composite in any LBA.
2. **For the subscribing and publishing composites, respectively, do the following:**
   - **Add `<Subscribing product> if not included in LBA short name.** Use the product short name as defined in Oracle Fusion Setup. Use `B2B` if the subscriber is a composite deployed to the Setup SOA cluster for B2B.
   - **Add `<Publisher product>To<Subscriber product>` so that the name is self-descriptive.** Use the product short name as defined in Oracle Fusion Setup.
3. **Identify the purpose of the composite.**
   - For a subscribing composite, add `XFamilySub` to the name.
   - For a publishing composite, add `XFamilyPub` to the name.
4. **End with Composite.**

### `XFamilySub`

A product can have more than one `XFamilyPub` composite.

Using the naming conventions specified in Composite Naming Conventions, the sample composite name is as follows:

- **`FinInfrZxXFamilySubComposite`.** In this example, the LBA short name is `FinInfr` and the `Zx` product subscribes to the cross-family event.
XFamilyPub

A product can have more than one XFamilyPub composite. There should only be one XFamilyPub composite for each publisher product and subscriber product combination.

Using the naming conventions specified in Composite Naming Conventions, sample composite names are as follows:

- **FinInfrHzToZxXFamilyPubComposite**. In this example, the LBA short name is FinInfr and the Zx product subscribes to an event published by the Hz product.
- **FinInfrPerToZxXFamilyPubComposite**. In this example, the LBA short name is FinInfr and the Zx product subscribes to the event published by the Per product.

### 47.4.3 Determining the Aqueue Message Recipient

A recipient or consumer is specified in the Aqueue Adapter defined in the XFamilyPub and XFamilySub composites. The recipient in the XFamilyPub is used to indicate which Aqueue Adapter should receive the enqueued message. The consumer in the XFamilySub is used to identify the Aqueue Adapter. For example, if the XFamilyPub specifies ZX as the recipient for the enqueued message, then the XFamilySub with the consumer set to ZX dequeues the message from the aqueue.

The recipient is the name of the XFamilyPub composite without the suffix XFamilyPubComposite, in the format <LBA short name><Publisher product>to<Subscriber product>.

For example:

- FinInfrHzToZx for FinInfrHzToZxXFamilyPubComposite
- FinInfrPerToZx for FinInfrPerToZxXFamilyPubComposite
- PrcInfrPoToB2B for PrcInfrPoToB2BXFamilyPubComposite

### 47.4.4 Defining an XFamilyPub Composite

This step defines the composite which subscribes to an event on a remote family’s Oracle SOA Suite cluster and enqueues a corresponding message in the ACR_XFAMILY_EVENT_Q aqueue.

1. In the owning team’s SOA workspace, create an empty composite with the name determined in Determining the Composites to Be Defined.

2. Drag and drop an Aqueue Adapter to the right swim lane of your composite and define the adapter as follows.

   a. In the Service Name field, enter a service name for the Aqueue Adapter.

   b. Define a connection for your Oracle Fusion Middleware schema and enter JNDI name eis/AQ/XFamilyEventAqueueInterface. In the Interface field, define the operation and schema.

   c. For the operation type, select the Enqueue radio button. In the Operation Name field, enter Enqueue.

   d. For the queue name, enter FUSION as the database schema and ACR_XFAMILY_EVENT_Q for the queue name.
e. On the Queue Parameters page, enter the recipient as defined in Determining the AQueue Message Recipient.

f. On the Message page, enter `oramds/apps/oracle/apps/common/acr/events/FusionXFamilyEvent.xsd` for the URL and `FusionXFamilyEvent` for the schema element. To access `FusionXFamilyEvent.xsd` from MDS, take the following steps.

   Click the search icon next to the URL field.
   
   In the Type Chooser dialog box, select Import Schema File.

   Click the search icon next to the URL field.

   At the top of the SOA Resource Browser window, select the Resource Palette.

   Under IDE Connections, navigate to `SOA-MDS > SOA-MDS connection Name > oracle > common > acr > events > FusionXFamilyEvent.xsd` and click OK.

   In the Import Schema File window, uncheck Copy to Project and click OK.

   In the Type Chooser window, select `FusionXFamilyEvent` and click OK.

3. Drag and drop a mediator to your composite and name it `EnqueueMediator`. Select Define Interface Later as the template and click OK. Double-click the `EnqueueMediator` component.

4. For each event to which your family subscribes, define a routing rule in the mediator. Click add button next to Event Subscriptions and select the business event to which you want to subscribe on the remote Oracle SOA Suite cluster. Reference the EDL file from MDS, and do not copy the file in your composite.

   a. Add a static routing rule and click the Service button. Select References > EnqueueEventMessage > Enqueue.

   b. Create a transformation XSL file by clicking the transformation icon in the routing rule and selecting Create New Mapper File.

   c. In the column on the right, right-click `msg_out:Namespace`. Select Set Text > Enter Text, and specify the event namespace.

   d. In the column on the right, right-click `msg_out:LocalName`. Select Set Text > Enter Text, and specify the event local name.

   e. In the Component Palette, from XSL Constructs, drag and drop copy-of to the `msg_out:Payload` element on the far right column of the XSL design tab.

      In the Copy-of Type dialog, select Replace the children of the selected node with the results of the copy-of. If an error message displays to the right of the copy-of element, you can ignore it.

   f. In the far left column, select the element just below `<sources>`. Drag-and-drop the element to the copy-of in the far right column in the XSL design tab. After you complete this step, the error icon that displayed in the previous step disappears.

      Example 47-1 shows a sample XSL file.

5. In the routing rule, define an assign statement to copy the context from the `apps.context.header` property into the `ApplicationContext` element in the aqueue message. The Assign Value window is shown in Figure 47-1.
Use the Expression builder to generate an expression for the To region. Select Variables > out > FusionXFamilyEvent > msg_out:FusionXFamilyEvent > msg_out:ApplicationContext, click Insert into Expression and then click OK.

**Figure 47-1  Define an Assign Statement**

The mediator source code is shown in Example 47-2.

6. Open the composite.xml file and search for the following line.

```xml
<binding.jca config="EnqueueEventMessage_aq.jca"/>
```

7. Replace the `binding.jca` element with the code shown in Example 47-3.

8. Add standard fault policies to the composite just above the `<component>` elements in the composite.xml file, as shown in Example 47-4.

**Example 47-1  Sample XSL File**

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<oracle-xsl-mapper
<!-- SPECIFICATION OF MAP SOURCES AND TARGETS, DO NOT MODIFY. -->
<mapSources>
  <source type="XSD">
    <schema location="../xsd/bulkImportEvents.xsd"/>
    <rootElement name="batchInfo"namespace="http://xmlns.oracle.com/apps/crm/hz/bulkImport/FoundationBulkImportEventsComposite"/>
  </source>
</mapSources>
<mapTargets>
  <target type="WSDL">
    <schema location="../EnqueueEventMessage.wsdl"/>
    <rootElement name="FusionXFamilyEvent"namespace="http://xmlns.oracle.com/apps/common/acr/events"/>
  </target>
</mapTargets>
<!-- GENERATED BY oracle XSL MAPPER 11.1.1.2.0(build 100216.1000.2230) AT [MON APR 26 14:00:00 PDT 2010]. -->
```
Example 47-2  Mediator Source Code

<assign>
</assign>

Example 47-3  Replace the JCA Binding

<binding.jca config="EnqueueEventMessage_aq.jca">
<property name="jca.subject.xpath"/>msg_out:FusionXFamilyEvent/msg_out:Subject</property>
<property name="jca.subject.nslist"/>msg_out="http://xmlns.oracle.com/apps/common/acr/events"/>
</binding.jca>
Example 47-4 Add Fault Policies to the composite.xml File

```xml
<property name="oracle.composite.faultPolicyFile">
  oramds:/apps/oracle/apps/fnd/applcore/soa/fault/fault-policies.xml
</property>
<property name="oracle.composite.faultBindingFile">
  oramds:/apps/oracle/apps/fnd/applcore/soa/fault/fault-bindings.xml
</property>
```

47.4.5 Defining an XFamilySub Composite

This step defines the composite which will dequeue messages from ACR_XFAMILY_EVENT_Q and raise the event locally in your family's SOA cluster.

**To define an XFamilySub composite:**

1. In the owning team's SOA workspace, create an empty composite with the XFamilySub composite name determined in Determining the Composites to Be Defined.

2. Drag and drop an Aqueue Adapter onto the left swim lane of your composite and provide the following values in the definition. When specifying the Messages URL, you will be referencing oramds. Ensure you have a SOA-MDS defined which points to fusionapps/soa_shared/soa-infra before proceeding.
   
   a. In the Service Name window, enter the name DequeueEventMessage in the Service Name text field.
   
   b. In the Service Connection window, define a connection for the Oracle Fusion Middleware schema. For the JNDI name, enter eis/AQ/XFamilyEventAqueue. For the XA Data Source, accept the default value.
   
   c. In the Adapter Interface window, accept the default selection Define from operation and schema (this is specified later).
   
   d. In the Operation window, select Dequeue for the operation type. In the Operation Name text field, accept the default value of Dequeue.
   
   e. In the Queue Name window, select the database schema and enter a queue name.
      From the Database Schema dropdown list, select Fusion.
      In the Queue Name field, enter ACR_XFAMILY_EVENT_Q.
   
   f. In the Queue Parameters window, specify the aqueue message recipient as defined in Determining the Aqueue Message Recipient.
   
   g. In the Messages window, select the message schema URL and the relevant schema element.
      In the URL field, browse for the message schema URL.
      From the Schema Element dropdown list, select FusionXFamilyEvent.

3. Drag and drop a mediator component onto your composite using the Define Interface Later template, and name it DequeueMediator.

   Draw a wire between the aqueue adapter and mediator, as shown in Figure 47-2.
4. For each cross family event to which your family subscribes, define a static routing rule in the mediator.

   a. Add a routing rule to raise the event by clicking on the add icon and the Event button. Use oramds to reference the EDL file rather than copying the EDL file into your composite.

   b. Add a filter expression to the rule to select the aqueue messages with the namespace and local name that match the event.

For example, if the namespace of the event is http://xmlns.oracle.com/apps/crm/hz/bulkImport/FoundationBulkImportEventsComposite and the local name is ImportPartyData, then the following filter is defined, as shown in Example 47-5.

   c. Create a transformation XSL file. Click the transformation icon, select Create New Mapper File, and click OK.

   d. In the Component Palette, expand the XSLT Constructs tab. Drag and drop the copy-of construct onto the XSLT file column on the element directly under the <target> element.

   e. In the Copy-of Type dialog, select Replace the selected node with the results of the copy-of.

   f. From the source column, drag and drop the msg_out:Payload onto the copy-of element in the XSLT file column.

   g. In the source tab, add a /child:node() element after <xsl:copy-of select="/msg_out:FusionXFamilyEvent/msg_out:Payload">.

The XSL file should look similar to that shown in Example 47-6.

   h. In the routing rule, define an assign statement to copy the context from the ApplicationContext element in the aqueue message into the apps.context.header property, as shown in Figure 47-3.
5. In the composite.xml file, replace the line `<binding.jca config="EnqueueEventMessage_aq.jca"/>` with the code shown in Example 47-7.

6. Add standard fault policies to the composite, as shown in Example 47-8.

**Example 47-5  A Filter Expression for the Mediator Rule**

```xml
and
($in.FusionXFamilyEvent/msg_out:FusionXFamilyEvent/msg_out:LocalName = "ImportPartyData")
```

**Example 47-6  XSL File**

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<oracle-xsl-mapper
<!-- SPECIFICATION OF MAP SOURCES AND TARGETS, DO NOT MODIFY. -->
<mapSources>
    <source type="WSDL">
        <schema location="../DequeueEventMessage.wsdl"/>
        <rootElement name="FusionXFamilyEvent" namespace="http://xmlns.oracle.com/apps/common/acr/events"/>
    </source>
</mapSources>
<mapTargets>
    <target type="XSD">
        <schema location="../xsd/bulkImportEvents.xsd"/>
        <rootElement name="batchInfo" namespace="http://xmlns.oracle.com/apps/crm/hz/bulkImport/FoundationBulkImportEventsComposite"/>
    </target>
</mapTargets>
<!-- GENERATED BY ORACLE XSL MAPPER 11.1.1.2.0(build 100216.1000.2230) AT [MON APR 26 15:28:04 PDT 2010]. -->
```
Example 47-7  Replacing the binding.jca Element

<binding.jca config="EnqueueEventMessage_aq.jca">
  <property name="jca.subject.xpath">/msg_out:FusionXFamilyEvent/msg_out:Subject</property>
  <property name="jca.subject.nslist">xmlns:msg_out="http://xmlns.oracle.com/apps/common/acr/events"</property>
</binding.jca>

Example 47-8  Adding Standard Fault Policies

<property name="oracle.composite.faultPolicyFile">
  orams://apps/oracle/apps/fnd/applicore/soa/fault/fault-policies.xml</property>
<property name="oracle.composite.faultBindingFile">
  orams://apps/oracle/apps/fnd/applicore/soa/fault/fault-bindings.xml</property>

47.5 Securing the Design Pattern

47.6 Verify the Deployment

You can verify the deployment of the use case by confirming that the event was successfully raised by checking the Event Delivery Network database log.
47.6.1 How to Verify the Deployment of the XFamilyPub Composite

To verify the deployment of XFamilyPub composite:

1. Deploy the XFamilyPub composite to the SOA cluster which receives the cross family business event.

2. Raise the cross family business event. Before integrating with Oracle ADF, you can raise a business event by invoking the following PL/SQL code in the Oracle Fusion Applications schema.
   a. Replace the parameters to initialize_session with the appropriate GUID and user.
   b. Replace hcm_edn_publish_event with the relevant event name, such as fin_edn_publish_event or crm_edn_publish_event, and so on.
   c. Replace the business event content.begin with the relevant business event.

Example 47-9 shows sample PL/SQL code used to raise a business event.

3. Confirm the event was raised on the SOA cluster that raises the cross-family business event by navigating to http://<SOA SERVER>:<SOA PORT>/soa-infra/events/edn-db-log.

4. Confirm the event name space, name, identity, payload and context matches the event raised via the PL/SQL API.
   a. In Fusion Applications Control, verify that an instance of the composite is running.
   b. Verify that a message is being enqueued (ensure the XFamilySub composite is not deployed) by looking at the contents of the FUSION.ACR_XFAMILY_EVENTS_QT table. The body of the message displays in the USER_DATA column. For example:

   ```sql
   select user_data from fusion.acr_xfamily_event_qt;
   ```

Example 47-9    Raise a Business Event by Invoking the PL/SQL Code

```sql
fnd_global.initialize_session('F5867912D280FD03AD1198A55EFC6BF', 'Abraham.Mason');

hcm_edn_publish_event('/oracle/apps/hcm/users/publicModel/entity/events/edl/LdapRequestEO', 'ActiveLdapRequestEvent',
   xmlns:ns="/oracle/apps/hcm/users/publicModel/entity/events/edl/LdapRequestEO">'||
   '<name>ns:ActiveLdapRequestEvent</name>'||
   '<content>'||
   '<ActiveLdapRequestEventInfo xmlns="/oracle/apps/hcm/users/publicModel/ entity/events/schema/LdapRequestEO">'||
   '</ActiveLdapRequestEventInfo>'||
   '</content>'||
   '</business-event>'
```

47.6.2 How to Verify the Deployment of the XFamilySub Composite

You can use Fusion Applications Control to verify that the composite is running.

**To verify the deployment of XFamilySub composite:**

1. Deploy the XFamilySub composite to the SOA cluster that subscribes to the cross-family event.
   
   The composite should pick up the message in the aqueue.

2. In Fusion Applications Control, verify that an instance of the composite is running.

3. Confirm the event was raised on the SOA cluster that subscribes to the cross-family business event by navigating to http://<SOA SERVER>:<SOA PORT>/soa-infra/events/edn-db-log.

4. Confirm the event namespace, name, identity, payload and context matches the event raised via the PL/SQL API.

47.7 Troubleshoot the Use Case

If you encounter any issue with the ACR_XFAMILY_EVENT_Q queue, first run the following select statements in the FUSION schema to confirm the privileges for FUSION_RUNTIME are correct and to confirm that the queue is configured for enqueuing and dequeuing. You can use the FUSION_READ_ONLY user to execute these statements.

### 47.7.1 Privileges to FUSION_RUNTIME

To confirm the necessary privileges are granted to FUSION_RUNTIME, run the statement shown in Example 47-10.

The statement should return the following results. Ensure SELECT, UPDATE, ENQUEUE, and DEQUEUE privileges are returned, as shown in Example 47-11.

If any of these privileges are not defined in your environment, run the appropriate command listed in Example 47-12.
Example 47-10  Confirming Privileges Granted to FUSION_RUNTIME

SELECT GRANTEE, OWNER, GRANTOR, PRIVILEGE, GRANTABLE, TABLE_NAME FROM DBA_TAB_PRIVS
WHERE TABLE_NAME like 'ACR_XFAMILY_EVENT%';

Example 47-11  Results Returned

<table>
<thead>
<tr>
<th>GRANTEE</th>
<th>OWNER</th>
<th>GRANTOR</th>
<th>PRIVILEGE</th>
<th>GRANTABLE</th>
<th>TABLE_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUSION_RUNTIME</td>
<td>FUSION</td>
<td>FUSION</td>
<td>SELECT</td>
<td>NO</td>
<td>ACR_XFAMILY_EVENT_QT</td>
</tr>
<tr>
<td>FUSION_RUNTIME</td>
<td>FUSION</td>
<td>FUSION</td>
<td>UPDATE</td>
<td>NO</td>
<td>ACR_XFAMILY_EVENT_QT</td>
</tr>
<tr>
<td>FUSION_RUNTIME</td>
<td>FUSION</td>
<td>FUSION</td>
<td>ENQUEUE</td>
<td>NO</td>
<td>ACR_XFAMILY_EVENT_Q</td>
</tr>
<tr>
<td>FUSION_RUNTIME</td>
<td>FUSION</td>
<td>FUSION</td>
<td>DEQUEUE</td>
<td>NO</td>
<td>ACR_XFAMILY_EVENT_Q</td>
</tr>
</tbody>
</table>

Example 47-12  Granting Relevant Privileges

GRANT SELECT ON FUSION.ACR_XFAMILY_EVENT_QT TO FUSION_RUNTIME;

GRANT UPDATE ON FUSION.ACR_XFAMILY_EVENT_QT TO FUSION_RUNTIME;

execute DBMS_AQADM.GRANT_QUEUE_PRIVILEGE('ENQUEUE', 'ACR_XFAMILY_EVENT_Q', 'FUSION_RUNTIME', FALSE);

execute DBMS_AQADM.GRANT_QUEUE_PRIVILEGE('DEQUEUE', 'ACR_XFAMILY_EVENT_Q', 'FUSION_RUNTIME', FALSE);

47.7.2 Aqueue enabled for Enqueuing and Dequeuing

To confirm ACR_XFAMILY_EVENT_Q is enabled for enqueuing and dequeuing, run the statement shown in Example 47-13.

This statement should return the results shown in Table 47-3. Confirm that the ENQUEUE_ENABLED and DEQUEUE_ENABLED columns are set to YES for ACR_XFAMILY_EVENT_Q.

Table 47-3  Results When Confirming ACR_XFAMILY_EVENT_Q Is Enabled for Enqueuing and Dequeuing

<table>
<thead>
<tr>
<th>Owner</th>
<th>Name</th>
<th>QUEUE_E_TAKE</th>
<th>QID</th>
<th>QUEUE_E_TAKE_MAX</th>
<th>RETR_Y DELAY</th>
<th>ENQUEUE_ENABLED</th>
<th>DEQUEUE_ENABLED</th>
<th>RETENuevo</th>
<th>USER_COMMENT</th>
<th>NET_WORK_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUSION</td>
<td>ACR_</td>
<td>ACR_</td>
<td>3851</td>
<td>NORM</td>
<td>5</td>
<td>YES</td>
<td>YES</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>XFAM</td>
<td>XFAM</td>
<td>358</td>
<td>AL_QUE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ILY_</td>
<td>ILY_</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>EVEN</td>
<td>EVEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T_Q</td>
<td>T_Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FUSION</td>
<td>ACR_</td>
<td>ACR_</td>
<td>3851</td>
<td>EXCEPTION</td>
<td>0</td>
<td>NO</td>
<td>NO</td>
<td>0</td>
<td>exception queue</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>$_AC</td>
<td>$_AC</td>
<td>357</td>
<td>PTO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>R_XF</td>
<td>R_XF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AMIL</td>
<td>AMIL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Y_EV</td>
<td>Y_EV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ENT_</td>
<td>ENT_</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>QT_E</td>
<td>QT_E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If the queue is not enabled for enqueue or dequeue, run the command shown in Example 47-14.
Example 47-13  Confirming ACR_XFAMILY_EVENT_Q Is Enabled for Enqueuing and Dequeuing

select * from sys.dba_queues where queue_table like '%ACR_XFAMILY_EVENT%';

Example 47-14  Enabling the Queue for Enqueuing or Dequeuing

dbms_aqadm.start_queue(queue_name => 'ACR_XFAMILY_EVENT_Q');

47.7.3 AQ_INVALID_QUEUE_TYPE

If you encounter the error message, shown in Example 47-15, take the following steps.

This error indicates that FUSION_RUNTIME does not have the appropriate privileges to enqueue messages in the aqueue. Take the steps described in Privileges to FUSION_RUNTIME.

Example 47-15  AQ_INVALID_QUEUE_TYPE Error

{{Error during invoking 1-way operation "Enqueue" on target service "EnqueueEventMessage"
Exception occurred when binding was invoked. Exception occurred during invocation of JCA binding: "JCA Binding execute of Reference operation 'Enqueue' failed due to: AQ_INVALID_QUEUE_TYPE. Unable to obtain the correct queue type. Queue does not exist or not defined correctly. Drop and re-create queue.". The invoked JCA adapter raised a resource exception. Please examine the above error message carefully to determine a resolution.  }}

Troubleshoot the Use Case
Use Oracle Enterprise Scheduler to Invoke an Asynchronous Service

This chapter describes how to invoke an asynchronous web service from an Oracle Enterprise Scheduler asynchronous Java job packaged within an Oracle Enterprise Scheduler hosting application. The scheduled job asynchronously invokes a web service operation, and the web service responds with a callback message to an Oracle Enterprise Scheduler specified endpoint URL.

This chapter includes the following sections:

- Introduction
- Alternative Approaches
- Invoking an Asynchronous ADF Business Components Service from an Java Job
- Securing the Design Pattern
- Verifying the Design Pattern
- Troubleshooting the Use Case
- What You May Need to Know About the Design Pattern

48.1 Introduction

This pattern is recommended for cases in which an asynchronous scheduled Java job executed by a hosting application must invoke an external asynchronous web service operation (such as a SOA composite or Oracle ADF Business Components service operation) using a Java API for XML Web Services (JAX-WS) client.

A Java job is scheduled to be executed by its Oracle Enterprise Scheduler hosting application. Oracle Enterprise Scheduler runs the job, invoking the asynchronous Java job. The job logic uses a JAX-WS client to invoke an external asynchronous web service operation.

The web service operation executes asynchronously in the background. On completion, the web service sends a response to the Oracle Enterprise Scheduler asynchronous callback web service using a callback message.

Involved components:

- Oracle Enterprise Scheduler, which invokes the asynchronous Java job at the scheduled time.
- A Fusion hosting application which executes the asynchronous Java job implementation specified by the scheduled job. The Java job implementation runs when Oracle Enterprise Scheduler executes the scheduled job.
• An external, asynchronous SOAP-based web service (SOA composite or ADF Business Components service).

48.2 Alternative Approaches

Other approaches involve using some form of JAX-WS client or proxy to invoke an external web service operation with custom logic for generating message IDs, correlating callback messages, and so on. Business events can also be used with SOA composites, but not with ADF Business Components or other non-SOA web services.

Following are alternative, unsupported approaches to the use case:

• Invoke services asynchronously, without referencing the Oracle Enterprise Scheduler asynchronous callback web service. This approach is not supported, as using a callback handler external to Oracle Enterprise Scheduler involves using a non-standard job completion notification with Oracle Enterprise Scheduler.

• Invoke a one-way service. This approach is not supported for Oracle ADF Business Components services.

• Using business events to invoke the external web service operation. This approach does not use a direct web service invocation, and will not work for Oracle ADF Business Components web service without using some indirection. This approach is not supported.

WARNING:
These alternative approaches are not supported and should not be implemented.

48.3 Example

You can download a sample asynchronous Java job implementation using a JRF JAX-WS client for invoking asynchronous web service operations here.

48.4 Invoking an Asynchronous ADF Business Components Service from an Oracle Enterprise Scheduler Java Job

This pattern describes how to use web service addressing SOAP headers to ensure message ID generation that allows ESS request id correlation like callback message correlation.

The following code shows how to use a JAX-WS client and WS address SOAP headers to invoke an asynchronous web service operation. This code is part of the job logic of an Oracle Enterprise Scheduler asynchronous Java job.

The invoked asynchronous web service must respond with a callback SOAP message to the endpoint address specified in \texttt{wsa:ReplyTo} SOAP header.

The web service can specify the following URI in the \texttt{wsa:Action} header of the callback message, to indicate that the web service execution resulted in an error: \url{http://xmlns.company.com/oracleas/schema/oracle-fault-11_0/Fault}

Any other URI in the \texttt{wsa:Action} field of the callback message is regarded as success by the Oracle Enterprise Scheduler asynchronous callback web service).
Example 48-1  Invoking an asynchronous ADF Business Components service

...  
// Set up a JAX-WS web service proxy or dispatch.
// The following sample uses a dispatch mode client.
// Similar code can be used for a JAX-WS proxy.
AddressingFeature wsAddressing = new AddressingFeature(true, true);
Dispatch<SOAPMessage> dispatch  = dispatchService.createDispatch(
    portQName, SOAPMessage.class,
    Service.Mode.MESSAGE, wsAddressing);

// Construct SOAP message.
String reqSoapMsg = createSOAPMessageString(wsOp.getRequestMessage());
SOAPMessage dispatchRequest = createSOAPMessage(
    jrfMessageFactory, reqSoapMsg);

// Get RequestContext and set up SOAP headers.
Map<String,object> dispatchRequestContext = dispatch.getRequestContext();

// Initialize SOAPACTION HTTP header.
dispatchRequestContext.put(
    BindingProvider.SOAPACTION_USE_PROPERTY, Boolean.TRUE);
dispatchRequestContext.put(
    BindingProvider.SOAPACTION_URI_PROPERTY, operation);

// Generate message ID from RequestExecutionContext.
String messageId = reqExecContext.toIdString();
saveExternalId(requestId, messageId);

// Initialize the WS message SOAP header: Action, MessageId, To, ReplyTo.
AddressingHelper.setOutgoingAction(
    dispatchRequestContext, operation);
AddressingHelper.setOutgoingMessageId(
    dispatchRequestContext, messageId);
AddressingHelper.setOutgoingToAddress(
    dispatchRequestContext, endpointURLString);
dispatchRequestContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
    endpointURLString);

// Configure the wsa:ReplyTo address as the Oracle Enterprise Scheduler Service asynchronous
// callback web service URL.
String callbackAddress = http://essHost:essPort/ess-async/essasynccallback;
AddressingHelper.setOutgoingReplyToAddress(
    dispatchRequestContext, callbackAddress);
AddressingHelper.setOutgoingFaultToAddress(
    dispatchRequestContext, callbackAddress);

// Configure the relatesTo string as the request EXTERNAL_ID system property value,
// only if this is a follow-on operation such as Cancel.
String relatesTo = getExternalId(reqParams);
AddressingHelper.setOutgoingRelatesToMessageId(
    dispatchRequestContext, relatesTo);

// Invoke the proxy or dispatch operation.
dispatch.invokeOneWay(dispatchRequest);
...

// Save the messageId string as an EXTERNAL_ID system property on request using RuntimeService.
protected void saveExternalId(long requestId, String messageId)
    throws ExecutionErrorException  {
    RuntimeService runtime;
Securing the Design Pattern

To secure this pattern, add the Oracle Web Services Manager policies to both the external web service endpoint (ADF Business Components service) and the callback client.

Add corresponding policies to the web service client and service policies subjects associated with the server running Oracle Enterprise Scheduler. The web service client policy is used to invoke the asynchronous web service operation, while the WS-service policy is used by the Oracle Enterprise Scheduler asynchronous callback web service to process callback responses.

For more information, see Securing Web Services Use Cases.

Verifying the Design Pattern

Build and deploy the Oracle Enterprise Scheduler hosting application which invokes the asynchronous Java job definition and job implementation, using Oracle JDeveloper. Ensure that the build and deployment are successful.

Use Oracle Enterprise Manager Fusion Applications Control to verify the job definitions associated with the hosting application. Submit the asynchronous Java job and verify the external asynchronous web service operation invocation.

Troubleshooting the Use Case

The following suggestions help resolve issues that may arise when deploying or running this use case.

Deployment

Compile and deploy the application in JDeveloper, and watch the log at the bottom of the window. If the compilation or deployment fails, the failures and suggestions for resolving issues are displayed in the log.

If deployment succeeds, but the Oracle Enterprise Scheduler hosting application does not display in Fusion Applications Control, check the server's diagnostic log and console output for any exceptions, and resolve them.

Runtime
If faults occur when invoking the job, the Oracle Enterprise Scheduler server log and job log messages include information to help resolve any issues.

For diagnosing invoked SOA composites at runtime, you may use the Audit Flow view to diagnose the problem and correct your BPEL process, then redeploy the process.

48.8 What You May Need to Know About the Design Pattern

Make sure the asynchronous web service and its WSDL is accessible online before scheduling the Oracle Enterprise Scheduler asynchronous Java job which invokes the web service operation.
This chapter describes the components that developers use to secure Oracle Fusion applications and Web services by enforcing authentication and authorization. This chapter includes the following sections:

- Introduction to Securing Oracle Fusion Applications
- Authentication Techniques and Best Practices
- Authorization Techniques and Best Practices
- Learning More About Technologies Used to Secure Oracle Fusion Applications

### 49.1 Introduction to Securing Oracle Fusion Applications

Oracle Fusion Applications security consists of two main components, namely authentication and authorization. Authentication establishes the identity of the user. Authorization ensures that users only have access to resources to which they have been granted access.

When developing an Oracle Fusion application, it is necessary to ensure that authentication and authorization policies are properly enforced throughout the application. The implementation details may vary depending on the technology used in the application.

Additionally, for information about security and extending Oracle Fusion applications, see the *Oracle Fusion Applications Extensibility Guide for Developers*.

### 49.1.1 Architecture

Oracle Fusion applications are built on top of a set of internal and third-party technologies. This set of technologies defines what may be used to develop, build, package, and run Oracle Fusion applications. Each technology and component may have its own specific requirements for security implementation. The following figure depicts the components in the Oracle Fusion Applications security approach and the relationship of these components to the related technologies. For details about the technologies that implement security components, refer to *Services (OPSS) Security Framework* through `c__2798`. 
49.1.1.1 Oracle Platform Security Services (OPSS) Security Framework

OPSS security framework provides security to Oracle Fusion Middleware, including Oracle WebLogic Server, Oracle SOA Suite applications, Oracle WebCenter Portal, Oracle ADF applications, and Oracle Entitlements Server. OPSS is designed to be portable to third-party application servers. Developers can therefore use OPSS as the single security framework for both Oracle and third-party environments, thus decreasing application development, administration, and maintenance costs.

OPSS comprises Oracle WebLogic Server security and Oracle Fusion Middleware security. The following figure illustrates the layered architecture that combines these two security frameworks.
The previous figure depicts the various security components as layers. The uppermost layer includes the Oracle WebLogic Server and the Java applications running on the server; under it, is the layer consisting of APIs for Authentication, Authorization, Credential Store Framework, User and Role, and identity virtualization; the bottom layer includes the Security Service Provider Interface (SSPI) layer and the service providers. The bottom layer interacts with security data repositories, such as LDAP and database servers.

In addition to the list of providers shown in Figure 49-2, other providers include the role mapping and audit providers.

The Security Service Provider Interface (SSPI) layer is accessed through OPSS APIs and provides Java EE container security. It also provides resource-based authorization for the environment, thus allowing customers to choose their security model.

For more information about SSPI, see the Oracle Fusion Middleware Understanding Security for Oracle WebLogic Server guide.

**Related Links**

The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For more information about OPSS, see the Understanding Security Concepts section in the Oracle Fusion Middleware Securing Applications with Oracle Platform Security Services, in the Oracle Fusion Middleware Online Documentation Library.

**49.1.1.2 c__2791**

Oracle Web Services Manager (Oracle WSM) provides a policy framework to consistently secure Web services.
Oracle Fusion Applications make use of an Oracle WSM feature called global policy attachments (GPA). Using GPA, web service policies are not attached locally, but are specified at a global level. At runtime, components simply inherit the global policy and Oracle WSM enforces it.

Unlike local policy attachments (LPA), which need to be added at every Web service client and server, global policy attachment can be attached at a domain level. This makes it easy for the system administrator to have a uniform policy for all Web services across the domain.

Because certain scenarios exist in which GPA cannot be used, Oracle Fusion application developers may need to use local policy attachment (LPA) for Web services and Web service clients. Application developers may attach policies locally using JDeveloper to the clients and services.

Authentication and authorization are enforced on the services by Oracle WSM based on the policies attached either at the domain level or locally on the service. The policies determine how the client and service communicate. For the predefined policies the naming convention indicates the behavior of the policy.

For example, given a policy called:

```
oracle/wss11_saml_token_with_message_protection_client_policy
```

- oracle is the path of the policy,
- ws11 indicates the Web services standard,
- saml_token is the authentication token,
- with_message_protection indicates whether message protection is enabled,
- client_policy indicates the type of policy, server or client.

Related Links

The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For more information about Oracle WSM, see the Oracle Fusion Middleware Administering Web Services guide.

How Policies are Executed

When a request is made from a service consumer (also known as a client) to a service provider (also known as a Web service), the request is intercepted by one or more policy interceptors. These interceptors execute policies that are attached to the client and to the Web service. There are five types of interceptors (reliable messaging, management, WS-Addressing, security, and MTOM) that together form a policy interceptor chain. Each interceptor executes policies of the same type. The security interceptor intercepts and executes security policies, the MTOM interceptor intercepts and executes MTOM policies, and so on.

Policies attached to a client or Web service are executed in a specific order via the Policy Interceptor Pipeline, as shown in the following figure.
When the interceptor encounters a policy that deals with authentication or authorization, it delegates the task to OPSS. If the authentication using OPSS is successful, a security subject with the identity is established. Similarly, processing continues only if the authorization using OPSS for the established identity is successful.

For details about the policy interceptor that Web services in Oracle Fusion applications use, see Securing Web Services Use Cases.

49.1.1.3 Oracle ADF Components and ADF Security

Oracle ADF security framework is the provider of authentication and authorization services to Oracle Fusion Applications. Oracle ADF security is built on top of OPSS architecture, and enables declarative, permissions-based security policies that grant users access to ADF bounded task flows and top-level web pages that use ADF bindings.

By default, ADF application resources are protected and thus require the application developer to configure security policies that grant role-based access privileges to enable access by users. A security administrator combines the declarative security policies with the domain-level policy store and configures a domain-level identity store with application users and role memberships.

At runtime, authenticated users are conferred access rights to the ADF resources by a combination of their role membership and the permission grants they inherit from these role memberships.

Oracle ADF security and Oracle JDeveloper provide the tools to interact with the file-based identity and policy store, as well as the architecture to enforce the security policy definitions on ADF application resources.

Related Links

The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

49.1.1.4 Application Context and Application User Sessions

Application user sessions allow applications to store the security and application context for Oracle Fusion Applications. Session attributes contain common information such as the current user and the user's associated roles, current language, date and number formatting, as well as application-specific attributes.

A session is created after the security subject has been established. In the context of Oracle ADF, a session is created using a filter, while a context is created in Oracle SOA Suite using an interceptor. When a session is created, information about the user's associated roles are stored in the session.

Oracle Fusion Data Security relies on the security context in the session when deciding whether a user is allowed to access particular data. If the session is not established, the user cannot access any secured data.

Application user sessions are associated with pillars and, ideally, there should be only one session per pillar. In the case of Web services, if the client and server are on the same pillar then they share the same session. Subsequently, session context is specific to a particular pillar. That is, everything running on that same pillar should see the same context.

For details about application user sessions, see Implementing Application User Sessions.

49.1.1.5 Oracle Fusion Data Security

Oracle Fusion Data Security implementation is a solution specifically for Oracle Fusion Applications. The security policy definitions for who can access what data are stored in the Oracle Fusion Data Security repository in the Oracle Fusion Applications schema. Oracle Fusion application developers who create new business objects can customize these definitions using Oracle Authorization Policy Manager, which can be accessed by the developer through the Setup and Maintenance work area of any Oracle Fusion Setup application.

Additionally, Oracle Fusion application developers must opt into the Oracle Fusion Data Security definitions in their application. In the context of an Oracle Fusion application, developers who want to require users to have sufficient data privileges can do the following:

- Declaratively enforce security on the entity object level
- Declaratively enforce security on the view object level through view criteria
- Programmatically apply security view criteria
- Programmatically call APIs to check whether a user is authorized to access data or obtain a predicate used to retrieve the data the user is authorized to access

The data security implementation relies on the security context defined in the application user session. Even if OPSS authenticates the user and the security subject is established, the user cannot access any data unless the application user session is established.

For details about Oracle Fusion Data Security, see Implementing Oracle Fusion Data Security.
49.1.1.6 c__2796

Oracle Virtual Private Database (VPD) enables controlling access to data on the database level using security policies associated with database objects. VPD controls access to rows and columns by adding a dynamic WHERE clause to a SQL statement issued against a database object to which an VPD security policy has been applied.

VPD can be useful for enforcing security when a development team must enforce security at the database level. Use VPD when securing Personally Identifiable Information (PII) on the database level, for example.

Using VPD affects performance. As such, make sure to evaluate your performance requirements before implementing the VPD solution.

Related Links

The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For more information about VPD, see the "Using Oracle Virtual Private Database to Control Data Access" chapter of the Oracle Database Security Guide, in the Oracle Fusion Middleware Online Documentation Library.

49.1.1.7 c__2798

Oracle Data Integrator (ODI) is used to move and transform data among systems using specific features for authentication and authorization. Authentication is based on Oracle Platform Security Services (OPSS). Following authentication, processing specific to ODI occurs in which OPSS principals are mapped to definitions stored in ODI to determine identity access rights. The security definitions controlling authorization decisions are stored in an ODI master repository.

Related Links

The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For more information about ODI, see the "Understanding Oracle Data Integrator" part in the Oracle Fusion Middleware Developing Integration Projects with Oracle Data Integrator, in the Oracle Fusion Middleware Online Documentation Library.

49.1.2 Authentication

Oracle Fusion applications reside in containers that automatically handle authentication. The container intercepts all requests entering the system and ensures that users are properly authenticated and the security context propagated.

Invoking the ADF Security wizard when developing an application configures the application to enforce security at runtime.

When a request is received with no subject defined, OPSS creates a subject containing the anonymous user principal and the anonymous role role principal. OPSS is configured by the JPSFilter. With this security subject, unauthenticated users can access public resources. When accessing secure resources, the adfAuthentication servlet forces users to authenticate. The security configuration determines the login module to be used for authentication.

By default, Oracle WebLogic Server is the authenticator used when developing applications with Oracle ADF. Different configurations can also be used, such as an Oracle Single Sign-On solution.
Related Links

The following document provides additional information related to Oracle Fusion Middleware topics discussed in this section:

- For more information about ADF Security, see the “Enabling ADF Security in a Fusion Web Application” chapter of the *Developing Fusion Web Applications with Oracle Application Development Framework*, in the Oracle Fusion Middleware Online Documentation Library.

### 49.1.2.1 Oracle Identity Management Repository

Oracle Identity Management Repository stores users, enterprise roles and their relationships. During development the identities exist in two places; the `jazn-data.xml` file in Oracle JDeveloper and the embedded LDAP of JDeveloper’s Integrated WebLogic Server. In staging environments, these definitions reside in LDAP on standalone Oracle WebLogic Server.

#### 49.1.2.1.1 Users

Test users created during development within Oracle JDeveloper enable testing applications in development. These test users are not migrated with the completed application. Rather, they are for testing purposes only. Enterprise users are added by a system administrator who defines users/groups in the enterprise identity store.

#### 49.1.2.1.2 Roles

Users are not assigned permissions directly, rather access is assigned to roles. Roles group particular permissions required to accomplish a task; instead of assigning individual permissions, roles match users with the permissions required to complete their particular task.

There are two main types of roles, enterprise and application. Oracle Identity Management Repository contains enterprise roles that are available across applications. These are created as groups in LDAP, making them available across applications. Application roles are stored in the application-specific policy store.

Functional roles include job, duty, data, abstract and privilege roles. Role are enforced by a role hierarchy. In Oracle Identity Management Repository, these logical roles are translated into technical Oracle Platform Security Services roles.

#### 49.1.2.1.3 Segregation of Duties

Segregation of Duties (SOD) ensures that no single individual has control over two or more phases of a business transaction or operation. The goal of segregation of duties is to prevent information misuse such that the same user cannot both create and approve transactions.

Oracle Applications Access Controls Governor (AACG) is used to manage, remediate and enforce user access policies. AACG ensures effective segregation of duties at the implementation site.

#### 49.1.2.1.4 File-Based Identity Store

The data in the file-based identity store (`jazn-data.xml` file) is used when authenticating within Oracle JDeveloper running Integrated WebLogic Server. The identity data in the `jazn-data.xml` file should not be synchronized with the identities in the LDAP staging server. By default, the deployment configuration disables data synchronization between the `jazn-data.xml` file and the LDAP server. This setting should not be modified.
49.1.2.1.5 File-Based Policy Store

The data in the file-based security policy store (jazn-data.xml file) is used when authorizing within Oracle JDeveloper running Integrated WebLogic Server. Changes to the security policies in the jazn-data.xml file must be migrated to the LDAP staging server by a security administrator. By default, the deployment configuration disables data synchronization between the jazn-data.xml file and the LDAP server. This setting should not be modified.

49.1.2.1.6 ODI

ODI has its own concept of identities such as ODI role and stores in the ODI schema. For authentication, OPSS is used and the OPSS principals are mapped to the ODI identities.

49.1.2.2 Identity Propagation

Identity propagation is a fundamental requirement for securing Oracle Fusion Applications. It provides that the same user identity is visible across different processes and technologies boundaries. While there some cases where the identity is implicitly propagated, in several scenarios explicit configuration is required.

Web Services and SOA

In the case of Web services and SOA applications, you can propagate identities by attaching Oracle Web Services Manager policies to the client and service. When a client sends a request to a service, a policy interceptor intercepts the request. On the client side, the policy interceptor packages the identity to be transported according to the policy attached to the client. On the service side, the interceptor processes the request based on the policy and delegates authentication to Oracle Platform Security Services. If the authentication succeeds, a security subject with the identity is established.

Remote Method Invocation (RMI)

The executing user is automatically propagated when using RMI. If a different identity than the executing user needs to be propagated, it has to be explicitly passed through Java Naming and Directory Interface (JNDI) context.

Oracle WebCenter Portal

Oracle Fusion applications deploy Web Services for Remote Portlets (WSRP). As Web services, they rely on Oracle Web Services Manager for Identity Propagation.

Oracle Data Integrator (ODI)

In a OPSS-enabled environment, ODI authentication happens in two phases.

The following figure depicts the first phase is OPSS authentication, during which time a subject is created using the OPSS framework. The second phase is the ODI authentication itself, which is based on the previously created subject. ODI lists the ODI roles having one or more OPSS principals of the subject as members. Users will need to choose one ODI role from here. ODI will then create an ODI security token based on this role. This ODI security token contains an ODI role and the list of principals associated with the current OPSS subject.
Application Identity

There are several Fusion Web application use cases in which privilege elevation is required. For example, when creating a new employee record, a new Trading Community Architecture (TCA) party record must be created. However, the identity of the user creating the employee has only employee creation privileges. A service must be invoked with temporarily elevated privileges to create the required party and complete the process. To enable privilege elevation, you can switch the identity of the invoker to a common identity with a higher level of access.

Audit Identity

When switching to the application identity, you must propagate the identity to be audited to preserve the submitting identity for auditing purposes.

Auditing is based on who columns, which are populated with session data. When the session is established, it is initialized using the current identity. The session APIs expose a method to manipulate the identity to be used for auditing. This method allows teams to control which identity is stored as the audit identity. The following example illustrates the syntax of this method.

The session is not propagated, rather only the identity is propagated. As the session is initialized using the identity, any application specific values are lost. To prevent this, you must pass the audit identity and override the audit identity on the session.

It is recommended for the service provider to add an extra parameter to the service so as to store the original user ID (historyOverrideUserName, of type String). In order to invoke the service, the service method consumer must fill in the original user ID as part of the payload. Within the service, the value passed is populated on the session as shown in the following example.
Example 49-1  Controlling Which Identity Is Stored as the Audit Identity

ApplSession.setHistoryOverrideUserName()

Example 49-2  Storing the Original Identity

ApplSession session = ApplSessionUtil.getSession();
if (session != null)
  session.setHistoryOverrideUserName(historyOverrideUserName);

49.1.2.3 Application User Session Propagation

Application user sessions are associated with pillars and, ideally, there should be only
one session per pillar. In the case of Web services, if the client and server are on the
same pillar then they share the same session. Subsequently, session context is specific
to a particular pillar. That is, everything running on that same pillar should see the
same context.

49.1.3 Authorization

Authorization ensures that users only have access to the resources to which they have
been granted access. Authorization decisions are based on policies stored in a policy
store. There are two main types of policy stores: OPSS application security repository
and Oracle Fusion Data Security repository. The OPSS repository contains the security
definitions that control access to applications. The Oracle Fusion Data Security
repository contains the security definitions controlling data access.

49.1.3.1 OPSS Application Security Repository

The enterprise security administrator exports the OPSS application security repository
to the jazn-data.xml file policy store. The security definitions in these repositories
control access to application functions. The policies defined in the jazn-data.xml
file are used during development. For testing and implementation, the file-based
policy store content is migrated into LDAP.

Policy Store Content

Enterprise security administrators are responsible for managing application security
policies. Oracle Fusion Applications developers can add new application security
policies, but must not modify existing application security policies.

Roles

The jazn-data.xml file identity store contains the application roles specific to a
given application. These roles are not visible outside the application. The policies are
created against an application role. Permissions are grouped into a permission sets for
administrative purposes. And permission sets are granted to the application roles.
Developers must not allowed to modify role hierarchy or remove privileges defined
by the permission sets granted to existing application roles.

Design Time

During development, you can interact with the jazn-data.xml file policy store
using the tools and user interfaces provided in Oracle JDeveloper.

Runtime

When Oracle ADF security is enabled in an application, the Web container uses the
policies in OPSS application policy store for authorization. Oracle ADF security
enforcement logic checks whether the user, represented by the JAAS subject, has the
correct permissions to access the resource.
The subject contains the user's principals, which include a user principal with the user's name and list of role principals, as well as enterprise roles and application roles obtained from the policy and identity stores. The principal is created to represent all of the user's memberships in application roles defined in the policy store. In turn, each application role may have multiple granted permissions in OPSS application policy store.

At runtime, the page context determines whether the current user has view permissions for the page being accessed. If the page includes an activity of a bounded task flow, the task flow controller determines the permissions. If the page is a top-level page with an associated page definition file, the Oracle ADF model determines the permissions for the page.

The OPSS service provider interface checks whether the subject includes the roles with the relevant permissions required to access the page. If the user is authorized to access the page, then the task flow is initiated. If the user is not authorized, ADF Controller throws an exception and passes control to an exception handler specified by the task flow configuration.

It is also possible to include an API that checks whether the current user has access to a resource.

Developer created additions to the policy store must be migrated to LDAP by a security administrator.

49.1.3.2 Oracle Fusion Data Security Repository

Oracle Fusion Data Security repository is used to control access to data.

Policy Store Content

Enterprise security administrators are responsible for managing data security policies. Oracle Fusion Applications developers can add new data security policies, but must not modify existing data security policies.

For more information about Oracle Fusion Data Security, see Implementing Oracle Fusion Data Security.

Design Time

During development, developers can interact with the Oracle Fusion Data Security repository through the Oracle Authorization Policy Manager.

Runtime

Data security relies on session information for the user identity. When a user session is created at runtime, the user information for that session and the flattened list of roles for the user are propagated to the database. This information is used to identify the user and the user's access level based on the policies in Oracle Fusion Data Security repository.

Data security is not automatically enforced, rather developers must enforce data security either declaratively on the entity object or view object, or programmatically, using API calls.

49.2 Authentication Techniques and Best Practices

There are some cases in which you must implement authentication from an external source or using a different identity. You can implement authentication using APIs, Expression Language or a non-browser based login.
49.2.1 API Support for Authentication

You can implement authentication by using user and role APIs.

**Related Links**
The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For more information, see the Developing with the User and Role API section of the *Oracle Fusion Middleware Securing Applications with Oracle Platform Security Services* guide, in the Oracle Fusion Middleware Online Documentation Library.

49.2.2 Expression Language Support for Authentication

You can use Expression Language (EL) to access security context information for authenticated users. Some useful expressions are as follows:

- `securityContext.userName`
- `securityContext.authenticated`
- `securityContext.userInRole`
- `securityContext.userInAllRoles`
- `securityContext.userGrantedPermission`
- `securityContext.userGrantedResource`
- `securityContext.taskflowViewable`
- `securityContext.regionViewable`

Note that decisions about user’s access rights should not rely on the user’s role information since role definitions may be changed. Instead access should be based on available permissions.

**Related Links**
The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For more information about ADF Security-based EL utility methods, see the "Enabling ADF Security in a Fusion Web Application" chapter of the *Developing Fusion Web Applications with Oracle Application Development Framework*, in the Oracle Fusion Middleware Online Documentation Library.

49.2.3 Non-browser Based Login

For information about using non-browser based security, see the "Securing Your Integrated Excel Workbook" chapter of the *Developing Applications with Oracle ADF Desktop Integration*, in the Oracle Fusion Middleware Online Documentation Library.

49.3 Authorization Techniques and Best Practices

Authorization is implemented using function security policies that control access to application functions. At the most fundamental level, authorization is based on standard JAAS (Java Authentication and Authorization Services) permissions and
OPSS permission sets (also called entitlements) which may be granted to secure specific application artifacts. Oracle ADF defines the JAAS permissions needed to secure certain Oracle ADF application artifacts, including ADF bounded task flows and, in the case of top-level web pages, ADF page definitions files.

49.3.1 Function Security

Security is automatically enforced on all ADF bounded task flows and top-level web pages that use ADF bindings and are not contained in a bounded task flow.

Security is not automatically enforced on Web service methods. You can use API calls to define permissions in the policy store and enforce security based on these permissions.

Implementing function security requires the following main steps:

1. Consult a security administrator to export all predefined function security policies of the application that you are customizing into a jazn-data.xml file.

2. Enable ADF Security for the application by running the Configure ADF Security wizard. This action updates the web.xml file with ADF Security configuration details and generates a temporary jazn-data.xml file.

3. Copy the exported jazn-data.xml file into the application workspace to overwrite the temporary file generated by the Configure ADF Security wizard.

4. Create an entitlement to group one or more ADF resources and their corresponding actions to entitle end users to access the resource.

5. Grant the entitlement to a custom duty role that was added to the Oracle Fusion application policy store.

After running the ADF Security wizard, any web page associated with a bounded ADF task flow will be protected. Therefore before running the application and testing security, developers must first create security policies that grant end users access.

For more information, see Implementing Function Security.

49.3.1.1 Resource Entitlements and Permissions

In general, the JAAS permission determines the allowed operations that the end user may perform on the application resource. However, from the standpoint of Oracle Fusion Applications, end users typically need to interact with multiple resources to complete the duties designated by their provisioned roles. To simplify the task of creating function security policies, developers work with entitlement grants (defined as OPSS permission sets) to grant privileges to access a variety of securable resources, including ADF task flows, Web services, and SOA work flows to a role.

Developers use the Oracle JDeveloper to create the entitlements (with one or more resource-action pairs) and then grant one or more entitlements to the desired application roles (the grantee).

For details about creating entitlement-based security policies, see Implementing Function Security.

Task flow, page definition, and Web service resource permissions are tightly coupled with code artifacts. These permissions are assumed to be associated with concrete code artifacts. In some circumstances, permissions are required, but no code artifacts exist with which the permissions could be associated. For example, suppose the same page is used to view and update tasks. The same code artifact is used for both actions such
that one cannot control access to both view and update tasks separately. Resource permissions enable creating abstract permissions which can be referred to with API calls.

Related Links
The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

• For details about resource permissions and using APIs, see the "Understanding Security Concepts" part in the Oracle Fusion Middleware Applications Security Guide, in the Oracle Fusion Middleware Online Documentation Library.

49.3.1.2 Expression Language Support for Authorization
You can use Expression Language to access security context information to determine the access rights of the authenticated user. Following are some useful expressions:

• securityContext.taskflowViewable
• securityContext.regionViewable
• securityContext.userGrantedPermission

Related Links
The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

• For more information about ADF Security-based EL utility methods, see the "Enabling ADF Security in a Fusion Web Application" chapter of the Developing Fusion Web Applications with Oracle Application Development Framework, in the Oracle Fusion Middleware Online Documentation Library.

49.3.2 Data Security
Implementing data security requires the following main steps:

1. Use Oracle Authorization Policy Manager to create security definitions based on the Oracle Fusion Data Security model. This step is common, regardless of the technology being used.

2. Refer to these security definitions from the code artifacts. This step varies depending on the technology used, as well as the functional requirements of the application.

For more information about implementing Oracle Fusion Data Security, see Implementing Oracle Fusion Data Security.

Related Links
The following document provides additional information related to Oracle Fusion Applications topics discussed in this section:

• For more information about using Oracle Authorization Policy Manager, see the "Using Oracle Authorization Policy Manager" chapter of the Administrator’s Guide.

49.3.2.1 API and Expression Language Support for Data Security
For information about using APIs and Expression Language to secure data, see Implementing Oracle Fusion Data Security.
49.3.2.2 Personally Identifiable Information

PII (Personally Identifiable Information) is any information that can be used to uniquely identify a person. This information is considered sensitive and must be protected from misuse for the purposes of legal regulation, financial liability and personal reputation. For example, only authorized users should be allowed access to the social security numbers of people stored in a system. PII authorization is only implemented on data identified by the PII working group in the data privacy Oracle Fusion uptake document.

PII authorization is implemented using one or more of the following technologies:

- OPSS Keystore APIs for encryption and decryption
- Oracle Fusion Data Security
- Oracle Virtual Private Database for row and column level security

The security requirements for the PII attribute determine the technologies to be used.

Related Links

The following document provides additional information related to Oracle Fusion Applications topics discussed in this section:

- For more information about OPSS encryption/decryption APIs, see the Oracle Fusion Middleware Securing Applications with Oracle Platform Security Services guide, in the Oracle Fusion Middleware Online Documentation Library.
- For more information about using VPD to control data access, see the Oracle Database Security Guide, in the Oracle Database Server Online Documentation Library.

49.3.2.3 Data Role Templates

Installations of Oracle Fusion Applications may require a large number of roles that must be provisioned. For data security purposes, it is often necessary to create the roles as the data security rules are not known at design time.

For more information regarding data role templates, see the product-specific security guides.

49.4 Learning More About Technologies Used to Secure Oracle Fusion Applications

The following resources within this book provide details about implementing Oracle Fusion Data Security and user sessions, and how to secure specific use cases for Oracle ADF application artifacts, Web services, and portlet applications.

- Oracle Fusion application user sessions. For more information, see Implementing Application User Sessions.
- Oracle Fusion Data Security policies for data security. For more information, see Implementing Oracle Fusion Data Security.
- Oracle Fusion Applications security policies for function security. For more information, see Implementing Function Security.
- Oracle WSM policy interceptors. For more information, see Securing Web Services Use Cases.

- Oracle Identify Management. For more information, see:

  Oracle Fusion Middleware Developing and Customizing Applications for Oracle Identity Manager.

- Oracle portlet services. For more information, see Securing End-to-End Portlet Applications.

Additionally, for information about security and extending Oracle Fusion applications, see the Oracle Fusion Applications Extensibility Guide for Developers.

**Related Links**

The following resources provide additional information about Oracle Fusion Middleware key security technologies and features used by Oracle Fusion Applications.

- Application resources including ADF task flows, ADF components, and ADF Business Components. For more information, see:

  Oracle Fusion Middleware Developing Fusion Web Applications with Oracle Application Development Framework, in the Oracle Fusion Middleware Online Documentation Library.

- Oracle Platform Security Services (OPSS) and identity providers (including the LDAP-based provider and the file-based provider (`jazn-data.xml` file)). For more information, see:

  Oracle Fusion Middleware Securing Applications with Oracle Platform Security Services, in the Oracle Fusion Middleware Online Documentation Library.

- Oracle Web Services Manager (Oracle WSM). For more information, see:

  Oracle Fusion Middleware Administering Web Services guide
Implement Oracle Fusion Data Security

This chapter describes how to use Oracle Fusion Data Security to enforce security authorization for access and modification of specific data records in an Oracle Fusion application.

This chapter includes the following sections:

- Introduction to Oracle Fusion Data Security
- Managing Data Security Artifacts in the Oracle Fusion Data Security Policy Tables
- Integrating with ADF Business Components
- Using Oracle Fusion Data Security to Grant Access to Secured Business Objects
- Getting Security Information from the Application User Session Context
- Understanding Data Security Performance Best Practices
- Validating Data Security with Diagnostic Scripts
- Integrating with Data Security Task Flows

50.1 Introduction to Oracle Fusion Data Security

Oracle Fusion Data Security is the technology that implements data security in Oracle Fusion Applications and is not used by function security (Oracle Platform Security Services (OPSS) is used for function security). Oracle Fusion Data Security integrates with Oracle Platform Security Services (OPSS) by granting actions to an OPSS principal. The grant defines who (the principal) can do what (the actions) on a given resource. A grant in Oracle Fusion Data Security can use any enterprise user or enterprise group as principals. Oracle Fusion Data Security assumes that the connection or session provided to it has been initialized properly with the appropriate user session user context, as described in Implementing Application User Sessions.

Note:

In this chapter, the user session is specifically an application user session (ApplCore). The application user session is the session that Oracle Fusion Data Security expects to see.

The goal of Oracle Fusion Data Security is to authorize a user to perform specified actions on selected data. It can secure rows and attributes of a database object and relies on OPSS to provide the authentication services for OPSS principals (users, groups, or roles). It answers the question "Who can do what on which set of data". 
**Who** refers to the OPSS user or group (or role), **what** is the action, and **which** is the subset of data that can be accessed.

You can use Oracle Fusion Data Security to either restrict the rows that are returned by a given query based on the intended business operation or you can restrict the actions that are available for a given row.

The purpose of data security is to model and enforce security authorization for a specific data record or a set of records. Data security provides access control within Oracle Fusion applications on the data a user can access and the actions a user can perform on that data. Oracle Fusion application rely on data security to restrict access to individual data that is displayed on a page that may display after the user has selected a menu or menu option.

The following are some use cases where Oracle Fusion Data Security can be utilized:

- Grant read action on expense reports to managers of the current employee when the manager is granted Expenses Administrator role.
- Administer the list of documents available to an end user in a Document Management System (DMS) based on Document Categories.
- Show the list of Sales Opportunities available to a Sales Head of an organization based on region.
- Allow a Human resources (HR) Benefits Administrator to only administer the employees whose last name begins with A-F.
- Allow a HR Administrator to only administer the employees in a given region.

In Oracle Fusion Data Security, data that must be secured is identified as resources. These resources are database tables or views. Policies that control which data that a user has access and can perform actions, can be made on a row instance or condition. The following figure illustrates the logical data model implemented by Oracle Fusion Data Security.

**Figure 50-1  Oracle Fusion Data Security — Logical Data Model**

An *instance* is a row of data and is identified by the primary key value of the row in the resource's storage table. A *condition* is a group of row instances whose membership is determined by a rule in the form of a SQL predicate, which must be applicable to the WHERE clause of a single-table query against the resource's storage table.

For example, each row in the Purchase Order table is an instance of the Purchase Order resource. The purchase order number is the primary key that identifies a particular purchase order instance. You can create an condition with the predicate "PO_NUMBER=100", which contains just one row of data. Purchase orders from the
West region can be put into a condition that is defined by the predicate "REGION='WEST'". A condition that contains all the rows of data in the resource's storage table can be defined by the predicate "1=1".

Memberships of a condition are dynamic in many ways, such as:

- Condition membership rules may contain any valid SQL attributes, such as columns of a table. Adding new instances or updating existing instances may affect the membership of a condition. Using the above Purchase Order example, if the predicate is "REGION='WEST'", new purchase orders in the region of West will automatically become a member of the condition.

- When an action is granted to an OPSS application role (also called a duty role by Oracle Fusion Applications) on a condition it can be parameterized. Using the Purchase Order example, the condition may be defined by the predicate "REGION=&PARAM" where the parameter PARAM is associated with different regions. When an action is granted on a condition, it may be done for a particular value of the parameter, such as a sales manager in the West region may have an action granted on a Region condition with the parameter value West.

- The condition rules may not reveal any membership at all. It can just be a WHERE clause to filter rows based on runtime user session variables.

To grant data security actions to a user, you must first identify the resources that you want to secure, define conditions on those resources, and then grant specific actions on these conditions to the application role to which the user belongs.

For additional information about Oracle Fusion Data Security, see the product-specific security guides.

Related Links

The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For additional information about OPSS, see the Oracle Fusion Middleware Securing Applications with Oracle Platform Security Services guide, in the Oracle Fusion Middleware Online Documentation Library.

50.1.1 Terminology

**Resource**: A resource on which data security is enforced, such as a purchase order. Resources are stored in the Oracle Fusion Applications FND_OBJECTS table. Note that Oracle Fusion Applications database tables are sometimes called FND tables, where FND refers to resources in the "foundation" tables.

**Instance**: A particular item of an resource, such as PO_NUMBER 100. An instance generally corresponds to a row in the database. Row instances have one or more primary key values.

**Condition**: A group of row instances that are determined by a SQL predicate (WHERE clause expression) that queries the attributes of the resource itself. The WHERE clause can reference values from the database context to implement relative conditions where the condition members depend on the security context of the current user. The conditions may also be parameterized, meaning that the WHERE clause references PARAMETER values from the policies for parameterized conditions, as described in How to Use Parameterized Conditions When Securing a Business Object.

Conditions are stored in FND_OBJECT_INSTANCE_SETS table.
**Action:** Secures an action (also called a function, privilege, or permission) that can be performed on a resource. You typically build features using multiple implementation strategies, including various ADF Business Components operations through Java code. These features must be secured to prevent unauthorized execution of the code. These features generally perform events on resources and actions and they are the artifacts used to secure these events. Actions are stored in `FND_FORM_FUNCTIONS` table, and the action names within the table are not unique. Only an action used in combination with a resource is unique.

**Aggregate Action:** A group of actions. Roles specify the combination of actions necessary to perform a particular role on a row instance. For example, a Project Administrator role may include the View, Update, Slip, and Delete actions and a Project Worker role may include only the View and Update actions. Aggregate actions (also called a menu) are stored in `FND_MENU` and `FND_MENU_ENTRIES` tables.

**Principal (Grantee):** A user or a role in Oracle Platform Security Services (OPSS) to which Oracle Fusion Data Security has a reference. The grantee key in the `FND_GRANTS` table holds the GUID of the OPSS user or role.

**User (OPSS User):** Any person or application that accesses information in the database.

**Role (OPSS Role):** Composed of users, groups, and possibly other roles. Roles are used to associate users with actions.

**Policy:** Authorization for the grantee (OPSS user or role) of an aggregate action may be done on the specified row instance, all instances, or condition. The condition for a policy may be static or parameterized. A policy logically joins a principal, aggregate action, and condition. This has the following effects:

- Any action granted on a row instance implies that the Oracle Fusion Data Security runtime system always has the ability to query the instances. This can be used by a standard Virtual Private Database (VPD) policy function to provide default query filtering. However, this does not mean that you have the ability to view or query because the ability to view a row of the resource is secured by an action.

- In the context of a specific row instance, the policies specify the set of actions that can be performed on a data record.

Resource access can be tested using the Oracle Fusion Data Security authorization checking API. Policies are stored in `FND_GRANTS` table.

**VPD - Virtual Private Database:** Provides the ability to dynamically attach a predicate at runtime to all queries issued against a database object (table or view). This feature is available in Oracle RDBMS. For more information about implementing VPD, see Integrating Oracle Fusion Data Security with Virtual Private Database (VPD).

**Security Policy Function:** A PL/SQL function developed to return a predicate added by VPD to a query. This function is bound to a table or view for some or all of DML statement types: SELECT, INSERT, UPDATE, DELETE.

### 50.1.2 Integrating Oracle Fusion Data Security with Oracle Platform Security Services (OPSS)

When integrating Oracle Fusion Data Security with Oracle Platform Security Services (OPSS) to support making policies to OPSS principals, it is important to understand that OPSS principals may be defined in third-party systems and this data does not exist in the database. Only the necessary information (user/user-role mapping) for the
current user session is propagated to the database at runtime during session creation to determine the various actions granted for that user session.

At runtime when a user session is created, the user information for that session and the flattened list of roles (to include role hierarchies) for the user of that session is propagated to the database. The roles available in a user session may be different from all the roles that a user may potentially have based on the authentication mechanism used, such as password vs. biometrics, authentication level of DMZ vs. non-DMZ, and so on. OPSS principal information is not stored in the Oracle Fusion Data Security schema.

50.1.3 Integrating Data Security Task Flows into Oracle Fusion Functional Setup Manager

Every Oracle Fusion application registers ADF task flows for setup activities with a product called Oracle Fusion Functional Setup Manager. These task flows are available from the Fusion Applications Setup and Maintenance work area and enable customers and implementers to set up and configure business processes and products. For more information about data security tasks, see the product-specific security guides.

If data security task flows are used in a web application, that web application must be configured to use ADF Security to enable authentication and authorization so that the correct data security predicates are generated.

Additionally, ADF Security controls access to a specific task flow, and users who do not have the required permission cannot view the task flow. For more information about how to implement function security permissions and roles, see Implementing Function Security.

The following table lists the task flows and their parameters.

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Parameters Passed</th>
<th>Behavior</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Database Resource</td>
<td>/WEB-INF/oracle/apps/fnd/appcore/dataSecurity/ui/taskflow/CreateDBResourceTF.xml</td>
<td>mode = edit dbResourceId</td>
<td>Goes to the Conditions tab of the database resource Edit page.</td>
<td>Conditions are a child entity of database resource. There is no Search page for conditions across all database resources; therefore, DB resource ID is mandatory.</td>
</tr>
<tr>
<td>Task Flow Name</td>
<td>Task Flow XML</td>
<td>Parameters Passed</td>
<td>Behavior</td>
<td>Comments</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------</td>
<td>------------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>Manage Database Resource Actions</td>
<td>/WEB-INF/oracle/apps/fnd/appcore/dataSecurity/ui/taskflow/CreateDBResourceTF.xml</td>
<td>mode = edit, dbResourceId, panelTab = actions</td>
<td>Goes to the Actions tab of the database resource edit page.</td>
<td>Actions are a child entity of database resource. There is no Search page for actions across all database resources; therefore, DB resource ID is mandatory.</td>
</tr>
<tr>
<td>Manage Policy</td>
<td>/WEB-INF/oracle/apps/fnd/appcore/dataSecurity/ui/taskflow/PolicyTF.xml</td>
<td>grantGuid (optional), dbResourceId (optional), appId (optional), roleName (optional)</td>
<td>This is the Create/Edit Policy page</td>
<td>There is no Search here, except to pick a specific database resource and pick a specific role.</td>
</tr>
</tbody>
</table>

### 50.1.4 Integrating Oracle Fusion Data Security with User Sessions

Oracle Fusion Data Security integrates with user sessions and relies on session context to be implemented properly.

For information about implementing user sessions, see Implementing Application User Sessions.

If a session has been created successfully, you will see the session created in the FND_SESSIONS table and the user session roles in the FND_SESSION_ROLES view.

When making policies to an OPSS principal, the GRANTEE_KEY must be a valid User / Role GUID as identified in the jazn-data.xml file. At runtime, the list of roles available to the user is determined by the roles granted to the user in the jazn-data.xml file and is populated in the FND_SESSION_ROLES view.

### 50.1.5 Integrating Oracle Fusion Data Security with Virtual Private Database (VPD)

Note that integrating with VPD is optional.

The database has a feature called Virtual Private Database (VPD). VPD allows an arbitrary WHERE clause to be appended to a table, view, or synonym. By doing so, the WHERE clause restricts the rows available. A PL/SQL function is written that returns the WHERE clause and a VPD policy is enabled on a particular table, view, or synonym that references that policy function. VPD policy functions based on fnd_data_security.get_security_predicate() are used to enforce data security rules.
Note:
If the base table holds PII data (personally identifiable information), the VPD policy must be created on the table and not on a view or synonym. If the policy is created only on a view or synonym, there is no protection against users accessing PII data from SQL*Plus. Setting the policy on the base table ensures that any PII data cannot be accessed by unauthorized users.

To integrate with VPD:
1. Create an action.
   Use Setup and Maintenance - Manage Data Security Policies UI to create an action on the database resource that you want to secure.

2. When the base table contains no PII data, create a view or synonym.
   If the table contains PII data, you will add the VPD policy directly to the table. Otherwise, create a view or synonym with the exact same name as the action.

3. Add a VPD policy to the table or view.
   Add a VPD policy in the database that will associate the policy function with the table or view.

   At runtime, in LOVs or UIs, wherever you want to display the rows that the user has select access to, they simply select off that view.

50.2 Managing Data Security Artifacts in the Oracle Fusion Data Security Policy Tables
Oracle Fusion Data Security artifacts include resources, row instances, conditions, actions, aggregate actions, and so on. Data security artifacts are stored in the Oracle Fusion Data Security repository and are customized using Oracle Authorization Policy Manager, which can be accessed by the developer through Oracle Fusion Functional Setup Manager, from the Manage Data Security task available in the Setup and Maintenance work area of any Oracle Fusion Setup application.

Note:
After the developer selects the Manage Data Security task in Oracle Fusion Functional Setup Manager, the environment redirects to the data security customization user interface provided by Oracle Authorization Policy Manager. In this document, although the data security customization tool is identified as Oracle Authorization Policy Manager, be aware that the tool must be accessed through Oracle Fusion Functional Setup Manager.

For details about creating and editing policies, see the "Managing Data Security Policies" topic in the Oracle Authorization Policy Manager online help. For supplemental task instructions, see the "Customizing Security for Oracle ADF Application Artifacts" chapter of the Extensibility Guide for Developers.
50.2.1 How to Get Started Managing Data Security

The user who logs in to view and manage database resources and policies must be authorized based on one of the roles described in Table 50-2.

Use cases:

- Application developer must be able to manage database resources for all Oracle Fusion resources.
- Product-family administrator must be able to manage only the database resources for their specific product family.

To use the standard Application Developer role:

The Oracle Fusion reference implementation predefines an Application Developer job role that inherits all the roles described in Table 50-2. You must contact a system administrator for Oracle Fusion Human Content Management (HCM) and request to be provisioned with the Application Developer job role. During the role provisioning process, you will be notified about the details of your account and password.

- Login to Functional Setup Manager with your account user name and password.

Alternatively, to use your own Product Family Administration role:

For example, if you have a CRM product family administration role, you must be able to manage the CRM database resources, but you will not be allowed to access the HCM resources.

If you have your own Product Family Administration role:

1. In Oracle Authorization Policy Manager, login as the user who is assigned to the appropriate duty role for your product family.

2. Inherit the appropriate duty role for your product family from the duty roles listed in Table 50-2.

3. Define the relationship between duty role and enterprise role.

How Oracle Authorization Policy Manager supports managing policies for database resources:

1. Oracle Fusion Applications delivers the duty roles and policies as specified.

2. Oracle Authorization Policy Manager authors enterprise roles and maps the duty roles listed in Table 50-2. A security manager uses Oracle Authorization Policy Manager to define the relationship between a duty role and enterprise roles.

3. A security manager can create their own enterprise roles if required. They can determine which database resources can be managed by specific data security administrators by including the duty roles into their custom enterprise roles. Security managers are expected to use Oracle Authorization Policy Manager to perform enterprise role to duty role mapping.

4. Policies runtime is based on role GUIDs only.

Reasoning:

Data security policies are granted to duty roles as the default approach. This makes it possible for the security manager to quickly assemble the duties to an enterprise role and use the Oracle Fusion reference implementation quickly. Granting them to an
enterprise role means that the security manager must duplicate the policies to any new enterprise roles they create. Enterprise roles are highly guarded and should be created and used only if absolutely needed.

50.2.2 What You May Need to Know About Administering Oracle Fusion Data Security Policy Tables

The data security administration UI is secured so that only administrators are permitted to create and manage security policies.

Note:
You cannot view the database resources or manage policies from the Functional Setup Manager if you have not been granted the appropriate data security manage permissions. Contact your security administrator for details.

The following table lists the duty roles that have been predefined by the Oracle Fusion security reference implementation to allow users to manage data security. These roles are administered at the product family level to manage resources and policies for that specific product family. The security reference implementation provides role-based access control in Oracle Fusion Applications, and is composed of predefined security policies that protect functions, data, and segregation of duties. Note that the table also identifies the duty roles associated with Oracle Authorization Policy Manager (APM) which allow developers to manage product family data security policies using the Manage Data Security Policies task.

Table 50-2 Duty Roles to Manage Data Security Policies

<table>
<thead>
<tr>
<th>#</th>
<th>Family</th>
<th>Duty Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CRM</td>
<td>Customer Relationship Management Database Resource Administration Duty</td>
</tr>
<tr>
<td>2</td>
<td>HCM</td>
<td>Human Capital Management Database Resource Administration Duty</td>
</tr>
<tr>
<td>3</td>
<td>FSCM</td>
<td>Financials and Supply Chain Manufacturing Database Resource Management Duty</td>
</tr>
<tr>
<td>4</td>
<td>APM - CRM</td>
<td>APM - CRM Database Resource Administration Duty</td>
</tr>
<tr>
<td>5</td>
<td>APM - HCM</td>
<td>APM - HCM Database Resource Administration Duty</td>
</tr>
<tr>
<td>6</td>
<td>APM - FSCM</td>
<td>APM - FSCM Database Resource Administration Duty</td>
</tr>
</tbody>
</table>

50.3 Integrating with ADF Business Components

In Oracle Fusion applications, the data to be secured is typically defined in the application’s data model project by an ADF Business Components entity object. Oracle Fusion Data Security integrates with ADF Business Components so that when you are defining an ADF Business Components entity object you can:

- Identify the actions that are available on a given row
- Have the ability to check at runtime if the user has access to this row based on the policies that are available to that user.
At runtime, authorization check is done automatically by ADF Business Components for standard operations that the Oracle Fusion application developer has declaratively enabled for business components, such as read, update, and removeCurrentRow. To perform a security check on non-standard operations you must call Oracle Fusion Data Security APIs directly.

Developers declaratively enable security for the desired operations on the ADF Business Components entity objects of their data model project. You may not identify operations directly on an ADF Business Components view object; however, when a view object references an entity object whose operations have been secured, the entity security policies also apply to the view object. In this regard, application developers must opt into security for individual business objects.

Oracle Fusion Data Security provides an implementation of a data security provider interface defined by ADF Business Components to perform the authorization check. The ADF Business Components data security provider interface provides a mechanism to filter rows based on a permission and to perform permission checks for certain actions. At runtime, Oracle Fusion Data Security uses an implementation of this interface to secure access to ADF Business Components entity objects and view objects.

50.3.1 How to Configure the ADF Data Model Project

To make the Oracle Fusion Data Security Provider as your data model project's data security provider, you can edit the Oracle Fusion application's adf-config.xml file to define the dataSecurityProviderClass attribute for the sec:JaasSecurityContext element, as shown in the following example.

For example, the adf-config.xml file would contain the sec:JaasSecurityContext element definition shown in the following example.

In Oracle JDeveloper, the design time tools for ADF Business Components are shaped so that the Oracle Fusion Data Security Provider will be automatically registered as the default when you launch JDeveloper with the Oracle Fusion Applications Developer role selected. This occurs when the developer runs the Configure ADF Security wizard for the ADF data model project.

At runtime, the ADF Business Components invocation of Oracle Fusion Data Security Provider happens automatically only for standard operations that have been enabled for entity objects of the data model project. For custom operations that are available on the entity object, you must invoke the Oracle Fusion Data Security authorization checking API manually, as described in How to Perform Authorization Checks for Custom Operations.

There may be other reasons to invoke Oracle Fusion Data Security APIs manually to determine the SQL predicate for a given action or to do an authorization check. For example, you might query VPD policies written based on fnd_data_security.getSecurityPredicate() to enforce data security rules.

Example 50-1 Making the Oracle Fusion Data Security Provider the Data Security Provider

dataSecurityProviderClass="oracle.apps.fnd.appcore.dataSecurity.util.FndDataSecurityProvider"

Example 50-2 sec:JaasSecurityContext Element Example

50.3.2 How to Secure Rows for Read, Update and Delete Access

At design time, you secure the read, update, and delete operations on a given entity object using Security section in the General page of the entity object overview editor. The overview editor exposes a set of standard operations (read, update, removeCurrentRow) as checkboxes that you can select. Until you select these operations, read, update, and delete access will not be enforced and therefore rows queried by entity-based view objects will remain accessible to users. Thus, to secure rows of data, the application developer must opt into security by enabling security for the standard row-level operations on the entity object, as shown in the following figure.

![Entity Object Overview Editor — Security Section](image)

To secure the entity object standard operations:

1. In the Application Navigator, double-click the entity object.
2. In the overview editor for the entity object, click the General navigation tab and expand the Security section.
3. In the Security section, select the operations that you wish to secure: read, update, or removeCurrentRow.

Note that the entity object permission class oracle.adf.share.security.authorization.EntityPermission defines the permissions read, update, and delete, which correspond to the operations read, update, removeCurrentRow, respectively.

50.3.3 How to Secure Rows for Custom Operations and Specific View Objects

Certain situations exist where it is preferable to secure operations at the level of view objects instead of a single entity object which protects access from all view objects derived from the entity object. In this sense, security at the level of view objects...
provides a more granular level of access control. Additionally, in cases where custom action checks needs to be performed, it is necessary to secure the view instance by applying a named view criteria defined for the purpose.

In the case of securing rows at the level of specific view objects, you use the View Criteria section of the General page of the overview editor for the view object to create the named view criteria. The following figure shows a view criteria that has been added in the overview editor for the view object.

**Figure 50-3  View Object Overview Editor — View Criteria Selection**

Once you have defined the view criteria, you must apply it to the view instance that you want the application module data model to filter, as described in the following procedure.

**To secure rows displayed to a user from a view object for a custom action:**

1. In the Application Navigator, double-click the view object that you will use to filter the rows.

2. In the overview editor, click the **Query** navigation tab and expand the View Criteria section, then click **Create New View Criteria** to add a view criteria.

3. In the Create View Criteria dialog, create a named view criteria with no view criteria items and name the view criteria using the following format:

   \[\text{FNDDS__permissionName__objectName__objectAlias}\]

   where:

   - **permissionName** is the permission name that is used to filter the data.
   - **objectName** is the name of the secured database resource in the **FND_OBJECTS** table.
   - **objectAlias** is optional and is the alias for the resource.
Note:
The delimiter is "__" (double underscore characters). This is because no other special character is allowed in a view criteria name.

4. Select **Both** for the **Query Execution Mode**, as shown in the following figure (note that no criteria items are defined).

   The query execution mode for the data security view criteria must be set to **Both**. If you leave the default setting **Database** selected, then the ADF Business Components association consistency feature will not work properly.

   ![Edit View Criteria Dialog](image)

   **Figure 50-4  Edit View Criteria Dialog**

5. In the application module overview editor, select the **Data Model** navigation tab and select the view instance to filter, then click **Edit** to apply the view criteria, as shown in the following figure.

   Alternatively, you can apply the view criteria at runtime in your code by calling the view object's `applyViewCriteria(viewCriteriaName)` API.
50.3.4 What You May Need to Know About Naming Actions

This means that when you define actions in Oracle Fusion Data Security, the actions for those business objects must be named as read, update, or delete to correspond to the entity object security operations that get enabled. Note that the action name corresponding to the removeCurrentRow operation is identified as delete by Oracle Fusion Data Security.

Similarly, when you create custom actions in Oracle Fusion Data Security, the action name that you define must match the custom action specified on the entity object.

50.3.5 What You May Need to Know About Column-Level Security

Oracle Fusion Data Security Provider only implements row-level authorization check. It does not implement a column-level authorization checking API. Even though Oracle Fusion Data Security can be used to perform column-level security using custom actions, it is not integrated with ADF Business Components directly using the data security provider interface. Wherever column-level security is needed, you must use a custom action.

50.3.6 What You May Need to Know About Translatable Entity Objects

The default Oracle Fusion Data Security provider implementation assumes that the object name in FND_OBJECTS for the entity object being secured is the database table/view name backing this entity object. If the entity object is a translatable entity object (MLS entity), then the backing database table/view name is identified by Oracle Fusion Middleware Extensions for Applications custom property fnd:OA_BASE_TABLE.
If the default behavior used to identify the backing database table/view name is not sufficient, one can use the entity object overview editor to set a custom property on the entity object. This custom property will identify the name of the object from the Oracle Fusion Data Security repository that will be used to secure this entity. The custom property OA_DS_BASE_TABLE should be set in the Custom Properties section in the General page of the entity object overview editor to accomplish this.

50.3.7 What Happens at Runtime: How Oracle Fusion Data Security Filters View Instance Rows

Based on standard operations that you enable for entity object security, the appropriate row-level permission checks are done at runtime. To enable these checks to happen automatically, JDeveloper registers the Oracle Fusion Data Security Provider with ADF Business Components in the adf-config.xml file when you secure an operation on the entity object.

At runtime, in the case of standard entity object operations that have been secured, ADF Business Components invokes the Oracle Fusion Data Security Provider in two steps:

1. The security provider first identifies the WHERE clause (if any) that must be added to the SQL statement for the entity object on which the read operation has been secured. This is done before executing the query.

2. After the query has been executed, ADF Business Components then invokes the Oracle Fusion Data Security Provider again to perform the authorization check for the update and removeCurrentRow operations to determine whether user has update and delete access to that row.

The implementation for securing data by applying a view criteria on the view object instance is handled in the Oracle Fusion Middleware Extensions for Applications layer. It requires the use of Oracle Fusion Middleware Extensions for Applications base classes to achieve this behavior. The OAViewCriteriaAdapter class is the default view criteria adapter class set as the ADF Business Components application module in the OAApplicationModuleImpl class. This functionality is provided in the OAViewCriteriaAdapter.getCriteriaClause() method to fetch the security predicate. In this implementation, the method uses the metadata on the view criteria to invoke Oracle Fusion Data Security APIs that fetch the security predicate.

In the case of a custom operation secured by a custom action, multiple data security view criteria can be applied to the view object. When multiple view criteria are applied, the WHERE clause corresponding to each view criteria is AND'ed. This is standard behavior for ADF Business Components. However, when a single data security view criteria for a given permission is applied, the instance sets corresponding to that permission are OR'ed. When multiple permission checks are applied, the instance sets of a permission are AND'ed with the instance sets of another permission. For example, for an object, for permission perm1, the user has instance sets IS1, IS2. For the same object for privilege priv2, the user has instance sets IS3, IS4. If both perm1 and perm2 checks are applied simultaneously (using view criteria), the WHERE clause would be (IS1 OR IS2) AND (IS3 OR IS4).

50.3.8 How to Perform Authorization Checks for Custom Operations

At runtime, the ADF Business Components invocation of Oracle Fusion Data Security Provider happens automatically only for standard operations. For custom operations that are available on the entity object, you must invoke the authorization check API manually as shown in the following example.
Note that the `allowsOperation()` method expects as input the secured operation name rather than the name of the action that secures it. In the case of the standard operations, the operation name for deleting an entity row is `removeCurrentRow` (where `delete` is the action name).

There may be other reasons to invoke Oracle Fusion Data Security APIs manually to determine either the SQL predicate for a given action or to do an authorization check. For example, VPD policies written based on `fnd_data_security.getSecurityPredicate()` to enforce data security rules.

**Example 50-3  Manually Invoking the Authorization Checking API**

```java
if(row.getSecurityHints().allowsOperation("ApprovePO").hasPermission())
    // code for P0 approval
else
    // display error message
```

### 50.3.9 How to Test Permissions Using Expression Language Expressions in the User Interface

You can test data security actions for standard operations (read, update, `removeCurrentRow`) or custom operations on an entity row using Expression Language or Groovy expression exposed on the entity row.

The Expression Language and Groovy expressions described below work by default when the view object is an updateable view object based on an entity object. If the view object is a read-only view object (based on an entity object) or an expert mode view object, then Expression Language or Groovy expression will not work unless you create a transient attribute with a custom Groovy expression that invokes Data Security to check the operation. The transient attribute can be used to control the rendering of some other attribute in the read-only view object on the page.

- The following example shows the Expression Language expression to use on ADF bindings for view object attributes.

  For example, your UI might have a button to control the grant for the `UpdateEmployeeSalary` custom action; the Expression Language expression on the button may be defined as shown in the following example.

  When this expression is invoked, Oracle Fusion Data Security Provider checks to see if the user identified by the `PersonId` attribute has access to the current row for `UpdateEmployeeSalary` operation. Note that even though the attribute name is included in the Expression Language expression, it is really doing a row-level security check.

  **Do not** use the expression shown in the following example as Oracle Fusion Data Security Provider does not implement a column-level security check interface. If you want to perform a column-level security, use the expression shown in the previous example with a custom action.

- The following example shows the expression to use for table iterators:

  **Do not** use the expression shown in the following example as Oracle Fusion Data Security Provider does not implement column-level security check interface. If you want to do column-level security, use the expression shown in the previous example with a custom action.

- When using Expression Language, be careful if you decide to set the expression on the `rendered` attribute. When PPR is enabled, ADF Faces does not handle the `rendered` attribute on a UI component well.
Tip:
PPR is enabled by default in Oracle Fusion Applications.

If you want to use Data Security expressions on the rendered attribute, you must manually identify the partial trigger UI components on the page and then set the partialTriggers attribute on the parent UI component of the UI component that has the data security Expression Language expression. Do not use visible attribute on the UI component as this could potentially be a security hole when the UI component and its data is rendered by the server and sent to the client. The visible attribute is a client-side attribute to show or hide the UI component on the browser.

- When using Groovy expressions, use the expression shown in the following example if the view object is an updateable view object based on an entity object.

- When using Groovy expressions, use the expression shown in the following example if the view object is a read-only view object based on an entity object or an expert mode view object. Create a transient attribute on the view object of type Boolean and Value Type Expression. The transient attribute can be used to control the rendering of some other attribute in the read-only view object on the page.

For example, you have a read-only view object with the attributes PersonId, Name, Gender, and Age and you want to control the rendering of the Gender attribute on the UI. First, you should create a transient attribute by name TransientGenderAttr and set the Groovy expression as mentioned above. The following example shows an EL expression to conditionally render the UI component for the Gender attribute:

You must make sure that the parent UI component of this UI component has the partialTriggers set to the appropriate UI component IDs. Also, in this scenario, make sure to have the binding available for the transient attribute in the view object, and to set the rendered="false" on the UI component for the transient attribute, or comment out the UI component in the JSF page.

**Example 50-4 Using Expression Language on View Object Attributes**

`#{bindings.<attrName>.hints.allows.<operationName>}`

**Example 50-5 Correct Expression Language Expression Example**

`#{bindings.PersonId.hints.allows.UpdateEmployeeSalary}`

**Example 50-6 Incorrect Expression Language Expression Example**

`#{bindings.<attrName>.hints.<attrName>.allows.<operationName>}`

**Example 50-7 Using Expression Language for Table Iterators Example**

`#{row.hints.allows.<operationName>}`

**Example 50-8 Incorrect Expression Language Expression Example**

`#{row.hints.<attrName>.allows.<operationName>}`

**Example 50-9 Groovy Expression Example**

`object.getSecurityHints().allowsOperation("updateCategory").hasPermission()`
Example 50-10  Groovy Expression if View Object is Read-Only Example

```java
testPrivilege("VIEW_PERSON_NAME LIKE_T","PER_EL_TEST_PERSONS", PersonId.toString(),
null,null,null,null,object.getViewObject().getDBTransaction());
```

Example 50-11  Implementing Attribute Security Example

```html
<af:panelFormLayout id="pfl1" partialTriggers="it6 cb3 cb1 cb4 cb5">
  <f:validator binding="#{bindings.Gender.validator}"/>
</af:inputText>

<af:inputText value="#{bindings.TransientGenderAttr.inputValue}" label="#{bindings.TransientGenderAttr.hints.label}" required="#{bindings.TransientGenderAttr.hints.mandatory}" columns="#{bindings.TransientGenderAttr.hints.displayWidth}" maximumLength="#{bindings.TransientGenderAttr.hints.precision}" shortDesc="#{bindings.TransientGenderAttr.hints.tooltip}" rendered="false" id="it1"/>
</af:inputText>

```
</af:panelFormLayout>
```

50.4 Use Oracle Fusion Data Security to Grant Access to Secured Business Objects

To secure the business object in the extended application, where it has been exposed as an ADF entity object, is a multi-step process:
1. A database resource definition must be created in the FND_OBJECTS table to identify the same table or view backing the ADF entity object. The database resource in Oracle Fusion Data Security is the data resource on which data security is enforced.

2. After the business object is defined as an Oracle Fusion Data Security database resource, then a security policy must be created to grant access to the data records. The security policies for the database resource specify access permissions such as read, update, and delete permissions on specific sets of data records exposed by the business object.

3. Finally, to enforce the newly created data security policy, the entity object in the data model project must enforce OPSS authorization checking on the data operations that access the protected data records exposed by the business object, as described Integrating with ADF Business Components.

As an Oracle Fusion Applications security guideline, a new data security policy must be created instead of modifying predefined data security policies of the security reference implementation. For example, a new data security policy is required to expose additional data records or operations for an existing business object. Additionally, a custom duty role must be created as the recipient of the new data security access permissions because granting permissions to a predefined duty role would alter the segregation of duties defined by the security reference implementation.

The general process for defining data security policies to grant access to business objects that you add to your Oracle Fusion application is as follows.

1. Identify the business object that you want to secure:
   a. Register your database view or table that you want to secure with Fusion Oracle Data Security.
   b. Populate the FND_OBJECTS and FND_OBJECTS_TL tables appropriately.

2. Identify the conditions that you want to make available on the registered business object:
   Tip:
   Conditions may be static or parameterized. For details about parameterizing conditions, see How to Use Parameterized Conditions When Securing a Business Object.
   - Populate the FND_OBJECT_INSTANCE SETS and FND_OBJECT_INSTANCE_SETS_TL tables appropriately.

3. Identify the actions that you want to secure this business object:
   - Populate FND_FORM_FUNCTIONS and FND_FORM_FUNCTIONS_TL tables appropriately.

4. Group the actions appropriately to form aggregate actions:
   a. Identify the name of the aggregate action and register it.
   b. Register the various actions that are part of the aggregate action.
   c. Compile the aggregate actions for faster reference.
You must do this to avoid hierarchical queries against the FND_Menu Entries table as aggregate actions may nest other aggregate actions. To compile the menus, invoke fnd_function.fast_compile or fnd_function.compile_all_from_scratch.

Note:
Menu hierarchies, such as sub-menus, are currently not supported. Menus may only include functions. This is because Seed Data Loaders do not support hierarchies now.

However, this step is still required as runtime queries are fired against the fnd_compiled_menu_functions table instead of the fnd_menu_entries table.

d. Populate FND_MENUS, FND_MENUS_TL, and FND_MENU_ENTRIES tables appropriately.

5. Identify the Oracle Platform Security Services (OPSS) principals (OPSS users and roles) for which you want to make policies.

OPSS principals do not exist in the database and are managed by OPSS Policy Store, which may be a third-party system.

6. Make appropriate aggregate action policies on the business resource to OPSS users and roles.

Policies can be made on a row instance, on the resource globally, or for a condition, which may be parameterized.

• Populate the FND_GRANTS table appropriately.

50.4.1 How to Create Data Security Policies for Custom Business Objects

Data security policies are stored in the Oracle Fusion Data Security repository and are defined and edited using Oracle Authorization Policy Manager. You have access to this tool through Oracle Fusion Functional Setup Manager, from the Manage Data Security task available in the Setup and Maintenance work area of any Oracle Fusion Setup application, however, you will need specific permissions. When you have the necessary permissions, Oracle Authorization Policy Manager allows you to access the data security customization user interface.

Note:
For complete details about Oracle Authorization Policy Manager tasks that you can perform with the Manage Data Security Policies user interface, see the "Customizing Security for Oracle ADF Application Artifacts" chapter of the Extensibility Guide for Developers.

Before you begin:
Contact your security administrator to receive the duty role membership that entitles you to perform the Manage Data Security Policies tasks in Oracle Authorization Policy Manager.
You must have configured your user interface project to use application user sessions to support the data security runtime. For more information about implementing application user sessions, see Configuring Your Project to Use Application User Sessions.

You must enable security on your application. For more information about enabling security, see How to Enforce Authorization for Securable ADF Artifacts.

Collect the following information that you will use to define the data security policy in Oracle Authorization Policy Manager:

- The primary key of the database table or view that the custom business object represents
  
  You specified the primary key of the database table or view when you registered the database resource in Oracle Authorization Policy Manager.

- The names of the conditions for which you want the security policy to control access
  
  When you registered the database resource, you may have created named conditions to control access to instance sets composed of multiple rows (Oracle Fusion Data Security does not require that you create a named condition when you want to grant access to instance sets composed either of a single row or of all rows of the database resource).

- The names of the actions for which you want to associate with a particular named condition (or instance set) to control access
  
  When you registered the database resource, you named actions to identify the securable operations of the custom business object. Action names must be identical to the names of the operations the business object supports. For example, the names of actions corresponding to the supported standard operations are view, edit, and delete. However, if your data model project defines custom operations, actions may have names corresponding to operations named, for example, as view_US_ONLY, edit_US_ONLY, or delete_US_ONLY.

- The names of the custom duty roles for which you want to grant access to the conditions and actions of the database resource associated with the custom business object
  
  As an Oracle Fusion Applications security guideline, predefined duty roles defined by the security reference implementation must not be modified. You must use Oracle Authorization Policy Manager to create a new duty role rather than grant data security permissions to predefined duty roles.

In Oracle Authorization Policy Manager, the general steps for defining a data security policy are as follows:

1. Using the Manage Data Security Policies user interface in Oracle Authorization Policy Manager, register the custom business object as a database resource.

2. Define the instance set of data records that you want to associate with specific securable operations of the ADF business component.
   
   The security policy identifies named conditions from the security repository to specify the row instance set available to the end user provisioned to the role with the permission to perform the intended ADF business component operation.

   In Oracle Authorization Policy Manager, a condition you create defines an instance set of multiple rows specified either by simple filters (XML-defined) or
complex SQL queries whose values can be parameterized. No condition definition is needed in the case of a single row instance or all the row instances of the database resource.

3. Define the list of actions that you want to be able to grant to the role.
Actions are database equivalent create, read, update, delete (CRUD) operations and correspond to the names of securable operations of the business object that the end user may invoke. The data security policy you define will associate one or more actions with an instance set.

4. If the custom business object is not supported by a data role template, define the data security policy:
   a. Enter a name and start date for the data security policy.
   b. Select one or more job roles or duty roles to which the policy grants access. The roles you select entitle all end users assigned to those roles with access to the data.
      In Oracle Authorization Policy Manager, duty role names that you enter are identified as OPSS internal roles called application roles. Similarly, job role names are identified as OPSS external roles called enterprise roles.
   c. Specify an instance set on the database resource for which the security policy will control access. This may be a single row, all rows, or multiple rows (specified by a previously defined named condition).
   d. Specify one or more actions to secure on the database resource for the currently specified instance set.
   e. Repeat the steps to grant actions access to additional instance sets for the current data security policy and roles.
      The following figure illustrates the Actions tab in the Edit Data Security page after several actions have been selected. Available actions will be limited to the actions that had been defined for the database resource.
Example: Securing a document categories business object

The following scenario describes the general steps that you would follow using Oracle Authorization Policy Manager to secure a document categories business object. For detailed information about the data security management tasks you can perform in Oracle Authorization Policy Manager, see the "Customizing Security for Oracle ADF Application Artifacts" chapter of the Extensibility Guide for Developers.

The document categories business data is stored in the `FND_DEMO_DOC_CATEGORIES` table. The following table lists the document category definitions for the table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATEGORY_ID</td>
<td>NOT NULL NUMBER</td>
</tr>
<tr>
<td>APPLICATION_ID</td>
<td>NUMBER</td>
</tr>
<tr>
<td>CREATION_DATE</td>
<td>NOT NULL DATE</td>
</tr>
<tr>
<td>CREATED_BY</td>
<td>NOT NULL NUMBER</td>
</tr>
<tr>
<td>LAST_UPDATE_DATE</td>
<td>NOT NULL DATE</td>
</tr>
<tr>
<td>LAST_UPDATED_BY</td>
<td>NOT NULL NUMBER</td>
</tr>
<tr>
<td>LAST_UPDATE_LOGIN</td>
<td>NUMBER</td>
</tr>
<tr>
<td>NAME</td>
<td>NOT NULL VARCHAR2(30)</td>
</tr>
<tr>
<td>START_DATE_ACTIVE</td>
<td>DATE</td>
</tr>
<tr>
<td>END_DATE_ACTIVE</td>
<td>DATE</td>
</tr>
</tbody>
</table>
The following example assumes that you have the necessary permission to perform the **Manage Data Security Policies** task in Oracle Authorization Policy Manager.

To secure a document categories business object in Oracle Authorization Policy Manager:

1. Create a resource named `FND_DEMO_DOC_CATEGORIES` and identify its primary key as `category_id`.

2. Create a static condition named `DMS_STATIC_INSTANCE_SET`, which secures categories 33, 34, and 35.
   
   The predicate for this condition is `'category_id in (33, 34, 35)'`.

3. Create a dynamic condition named `DMS_PARAMETERIZED_IS` to secure categories, which can be identified at grant time.
   
   The predicate for this condition is `'category_id in (%GRANT_ALIAS.PARAMETER1, %GRANT_ALIAS.PARAMETER2)'`, where `&GRANT_ALIAS` refers to the Policies table.

4. Identify the actions securing this entity object for *read*, *update*, and *delete* operations as read, update, delete. Additionally, custom actions with the names `FND_DEMO_ATTACHMENT_VIEW`, `FND_DEMO_ATTACHMENT_UPD`, and `FND_DEMO_ATTACHMENT_DEL` may be created for testing.

5. Identify aggregate actions for which policies are made for this entity:

   - Create an aggregate action named `FND_DEMO_ATTACHMENT_VIEW` for the *read* operation, which contains actions named read and `FND_DEMO_ATTACHMENT_VIEW`.

   - Create an aggregate action named `FND_DEMO_ATTACHMENT_ADMIN`, which allows a user to administer this resource. It has *read*, *update*, and *delete* actions. This aggregate action contains read, update, delete, `FND_DEMO_ATTACHMENT_VIEW`, `FND_DEMO_ATTACHMENT_UPD`, and `FND_DEMO_ATTACHMENT_DEL` actions.

6. Compile the menus in the database by executing:

   ```
fnd_function.compile_all_from_scratch;
   ```

7. Create OPSS principals.

   For this example the user names created are `joeUser` and `admin`. The role names created are `regularUserRole` and `admin`.

8. Make policies to OPSS principals `admin` and `regularUserRole` for the `FND_DEMO_DOC_CATEGORIES` business resource.

---

**Table 50-3 (Cont.) FND_DEMO_DOC_CATEGORIES Table Definition**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE_CATEGORY</td>
<td>VARCHAR2(30)</td>
</tr>
<tr>
<td>ATTRIBUTE1 thru ATTRIBUTE15</td>
<td>VARCHAR2(150)</td>
</tr>
<tr>
<td>DEFAULT_DATATYPE_ID</td>
<td>NUMBER</td>
</tr>
</tbody>
</table>

---

*Use Oracle Fusion Data Security to Grant Access to Secured Business Objects*
• Grant FND_DEMO_ATTACHMENT_ADMIN aggregate action on the resource globally to OPSS admin role.

• Grant FND_DEMO_ATTACHMENT_VIEW aggregate action to regularUserRole for static DMS_STATIC_INSTANCE_SET and dynamic (DMS_PARAMAETERIZED_IS with parameters 37 and 38) conditions.

50.4.2 How to Use Parameterized Conditions When Securing a Business Object

Parameterized conditions allow conditions to be specified generally but granted specifically. Parameterized conditions are useful because they reduce the number of predicates that the database must parse, as well as reducing the number of conditions that the administrator needs to manage.

The following example shows how a parameterized condition can be reused.

An administrator can reuse the first condition for several different policies granted to different locations and the second condition can be reused for policies granted to different titles. For example, one policy might use OIS1 to grant to 'WEST' by putting 'WEST' in FND_GRANTS.PARAMETER1, while another policy would reuse the same OIS1 to grant to 'EAST'. At runtime, the FND_DATA_SECURITY package substitutes the PARAMETER values from the FND_GRANTS table to the OISs granted.

When the data security system runs the predicates that have been defined in the conditions, it does a simple replace-style parsing of the predicate. For example, &TABLE_ALIAS is replaced by the table alias of the resource table, if it was passed to the get_security_predicate() call, and &GRANT_ALIAS is replaced by the policy table alias.

Caution:

The &TABLE_ALIAS and &GRANT_ALIAS column qualifiers must be included in the predicates of all conditions to keep possible duplicate column names from causing collisions.

Example 50-12 Reusing Parameterized Conditions

Condition. Employees in a particular region.
Predicate: "&TABLE_ALIAS.REGION = &GRANT_ALIAS.PARAMETER1"

50.4.2.1 Converting Non-String Parameter Values Into Character Values

The policy parameters hold only string values so you must convert non-string values into character values.

To convert non-string values into character values:

• integer- NUMBER with no decimal point. For example, 123.
  To store the number in the policy PARAMETER column, use to_char() without a format mask.

• float- NUMBER that can have a decimal point. For example, 123.456.
  This should be stored in the canonical format, using
  FND_NUMBER.NUMBER_TO_CANONICAL(). (The canonical format is based on the
  FND_NUMBER package
  'FM999999999999999999999.99999999999999999999'.)
You should use canonical format because the string must be stored in a format that doesn't need to be converted if the data is passed between systems that use different decimal characters or other number formatting.

- **date-** DATE calendar date, optional time. For example, 2009/04/30 11:32:32.
  
  This should be stored in canonical format, using FND_DATE.DATE_TO_CANONICAL(). (The canonical format is based on the FND_DATE package, 'YYYY/MM/DD HH24:MI:SS'.
  
  You should use canonical format because the string must be stored in a format that doesn't need to be converted if the data is passed between systems that use different date formats.

- **varchar2-** VARCHAR2 character string. For example, FND_THING_NAME.
  
  This can be stored without any conversion.
  
  However, translated values should not be stored here. This is for internal developer key values. There is no facility for having different multilingual values in different languages because predicates should not be comparing translated values.

### 50.4.2.2 Writing Performance Type Conversions in Predicates

Use the following rules when writing performance type conversions in your predicates.

**Integer Equality:**

The following example shows the correct Integer equality format to use.

The following example shows Integer equality formats that you should NOT use.

You must use to_char() as it is built-in and performs much faster than fnd_data_security.to_init().

When used with data security parameters, to_number() causes Invalid Number exceptions. This in turn, will make SQL statements abort if the to_number() gets run on policy parameters, which store non-numeric data such as DATE or VARCHAR2 data. The policies table will almost certainly contain some non-numeric data in a policy.

It may be decided to run a to_number(&GRANT_ALIAS.PARAMETER1) in a big statement on more policy rows than intended, and filter the rest of the rows later in the execution. This will definitely cause your SQL statement to fail so therefore, you must not have to_number() around any of the policy parameters in your predicate.

**Tip:**

The routine fnd_data_security.to_init() was written to avoid this problem; it is basically a wrapper over to_number(), which traps the Invalid Number exception. Therefore, if the execution plan involves operating against policies that don’t apply, they won’t cause the whole statement to fail. However, because of performance issues, to_char() is the preferred solution.

**Integer Range:**

The following example shows the correct Integer range format to use.

The following example shows Integer range formats that you should NOT use.
As explained previously, `to_number()` can fail with an *Invalid Number* exception if other policy rows are processed that have non-numeric parameters. `to_char()` does not produce the correct ordering. For example, '3' > '25'.

**Float Equality:**

The following example shows the correct Float equality format to use.

The following example shows Float equality formats that you should NOT use.

Using `to_number()` without a format could fail if the environment is set up to use comma as the decimal character and the parameter is stored with a period as the decimal character. Using `to_char()` without an explicit format could convert to a comma-format number, which would not match the period-format number. Because of these potential problems, you must explicitly provide the canonical format.

**Float Range:**

The following example shows the correct Float range format to use.

The following example shows Float range formats that you should NOT use.

You should not use `to_char()` because it does not order correctly.

You should use `fnd_data_security.to_decimal()` instead of `fnd_data_security.to_init()` because the data may contain decimals, which `to_init()` cannot handle.

**Date Equality:**

The following example shows the correct Date equality format to use.

The following example shows Date equality formats that you should NOT use.

As mentioned previously, `to_char()` performs best because it is built-in. The format mask is also required to make sure the canonical format is used.

**Date Range:**

The following example shows the correct Date range format to use.

The following example shows Date range formats that you should NOT use.

In this case, you can use `to_char()` with a format mask because the canonical format maintains proper ordering of the character domain.

**Example 50-13  Integer Equality Format**

```
      to_char(&TABLE_ALIAS.ITEM_ID) = &GRANT_ALIAS.PARAMETER1
```

**Example 50-14  Incorrect Integer Equality Formats**

```
&TABLE_ALIAS.ITEM_ID = to_number(&GRANT_ALIAS.PARAMETER1)
&TABLE_ALIAS.ITEM_ID = fnd_data_security.to_decimal(&GRANT_ALIAS.PARAMETER1)
```

**Example 50-15  Integer Range Format**

```
&TABLE_ALIAS.LEVEL >= fnd_data_security.to_init(&GRANT_ALIAS.PARAMETER1)
```

**Example 50-16  Incorrect Integer Range Formats**

```
&TABLE_ALIAS.LEVEL >= to_number(&GRANT_ALIAS.PARAMETER1)
to_char(&TABLE_ALIAS.LEVEL) >= &GRANT_ALIAS.PARAMETER1
```
Example 50-17  Float Equality Format

to_char(&TABLE_ALIAS.TEMP,'FM999999999999999999999.99999999999999999999') = &GRANT_ALIAS.PARAMETER1

Example 50-18  Incorrect Float Equality Formats

to_char(&TABLE_ALIAS.TEMP) = &GRANT_ALIAS.PARAMETER1
&TABLE_ALIAS.TEMP = to_number(&GRANT_ALIAS.PARAMETER1)
&TABLE_ALIAS.TEMP = fnd_data_security.to_decimal(&GRANT_ALIAS.PARAMETER1)

Example 50-19  Float Range Format

&TABLE_ALIAS.TEMP > fnd_data_security.to_decimal(&GRANT_ALIAS.PARAMETER1)

Example 50-20  Incorrect Float Range Formats

to_char(&TABLE_ALIAS.TEMP,'FM999999999999999999999.99999999999999999999') > &GRANT_ALIAS.PARAMETER1
&TABLE_ALIAS.TEMP > to_number(&GRANT_ALIAS.PARAMETER1)
&TABLE_ALIAS.TEMP > fnd_data_security.to_init(&GRANT_ALIAS.PARAMETER1)

Example 50-21  Date Equality Format

&TABLE_ALIAS.ACTION_DATE = fnd_data_security.to_date(&GRANT_ALIAS.PARAMETER1)

Example 50-22  Incorrect Date Equality Formats

to_char(&TABLE_ALIAS.ACTION_DATE) = (GRANT_ALIAS.PARAMETER1)
&TABLE_ALIAS.ACTION_DATE = to_date(&GRANT_ALIAS.PARAMETER1)
&TABLE_ALIAS.ACTION_DATE = to_date(&GRANT_ALIAS.PARAMETER1)

Example 50-23  Date Range Format

&TABLE_ALIAS.HIRE_DATE > fnd_data_security.to_date(&GRANT_ALIAS.PARAMETER1)

Example 50-24  Incorrect Date Range Formats

&TABLE_ALIAS.HIRE_DATE = fnd_data_security.to_date(&GRANT_ALIAS.PARAMETER1)
&TABLE_ALIAS.HIRE_DATE = to_date(&GRANT_ALIAS.PARAMETER1)

50.4.3 How to Create Test Users in JDeveloper

For development purposes, you can use the OPSS jazn-data.xml flat file to create users and roles for testing.

For an example of the identity store in the flat file used by JDeveloper, see the Enabling ADF Security in a Fusion Web Application chapter of the Developing Fusion Web Applications with Oracle Application Development Framework, in the Oracle Fusion Middleware Online Documentation Library.

50.4.4 What You May Need to Know About Creating Application Roles

Before testing the application in the staging environment, any custom application roles that you created will need to be created in the LDAP application policy store. These new application roles will receive new GUIDs and any data security policies defined for application roles of the same name must have their GUIDs reconciled. For details about reconciling GUIDs in the data security repository, see the Reconciling GUIDs section in the Administrator’s Guide.
50.5 Getting Security Information from the Application User Session Context

Oracle Fusion Data Security is a layer that interacts with Oracle Platform Security Services. Hence data security information can be obtained from security context as required while developing the application.

50.5.1 How to Use the DataSecurityAM API to Get Session Context Information

The DataSecurityAMImpl class is in the oracle.apps.fnd.applcore.dataSecurity.dataSecurityService.applicationModule package. It is the container for all data security resources provided by Oracle Fusion Middleware Extensions for Applications. This class contains core methods to:

- Check action for the current user, as shown in the following example.
  
  Where DataContext is a container class that represents the resource and optionally the primary keys of that resource for which data security check must be done. It has the following attributes: ObjectName, PK1, PK2, PK3, PK4, PK5.

```
Note:
Privilege and DataContext classes are in the oracle.apps.fnd.applcore.dataSecurity.util package.
```

If the action is granted to an Oracle Platform Security Services (OPSS) role in FND_GRANTS table, then the system retrieves all the OPSS roles that the user has access to, and checks if one of them matches with the OPSS role to which the action has been granted.

- Get the security predicate associated with a given action on a given resource for the current user, as shown in the following example.

- Get all the actions available on a given row instance for the current user, as shown in the following example.

- Get all the aggregate actions available on a given row instance for the current user, as shown in the following example.

As shown in the following example, static APIs are also provided in the DataSecurityAMImpl class for the above core methods that take in a DBTransaction object.

**Example 50-25 Check Action Method**

```java
/**
  * Test if an action is accessible for a given data context for the user
  * in the current session context.
  * @param privilege the privilege to test
  * @param dataContext data context (object and primary key) to use for
  * testing the privilege.
  * @return true if privilege is accessible, false otherwise.
  */
  public boolean testPrivilege(Privilege privilege, DataContext context)
```
Example 50-26  Get Security Predicate Method

/**
 * Return the security predicate for a given action on a resource
 * for the user in the current session context.
 * @param privilege privilege to use to determine the predicate.
 * @param dataContext the data context, which provides the object name.
 * @param grantInstanceType the grant instance type, (such as "UNIVERSAL").
 * @param statementType the statement type, (such as "OTHER").
 * @param tableAlias alias for the table.
 * @return the security predicate.
 */
public String getSecurityPredicate(Privilege privilege, DataContext dataContext,
        grantInstanceType, String statementType, String tableAlias)

Example 50-27  Get All Actions Available Method

/**
 * Provides the list of action names granted to the user for
 * the specified dataContext (Object/PK) in the current session context.
 * @param dataContext data context (object and primary key) to use for
 * identifying the granted actions.
 * @return list of action names.
 */
public String[] getPrivileges(DataContext dataContext)

Example 50-28  Get All Aggregate Actions Method

/**
 * Provides the list of aggregate action names granted to the user for
 * the specified dataContext (Object/PK) in the current session context.
 * @param dataContext data context (object and primary key) to use for
 * identifying the granted aggregate actions.
 * @return list of aggregate action names.
 */
public String[] getAggregatePrivileges(DataContext dataContext)

Example 50-29  Static API Examples

public static boolean testPrivilege(Privilege privilege, DataContext context,
        DBTransaction dbTxn)

public static String getSecurityPredicate(Privilege privilege, DataContext dataContext,
        grantInstanceType, String statementType, String tableAlias, DBTransaction dbTxn)

public static String[] getAggregatePrivileges(DataContext dataContext, DBTransaction dbTxn)

public static String[] getPrivileges(DataContext dataContext, DBTransaction dbTxn)

50.5.2 How to Use the PL/SQL Data Security API to Check User Privileges

Oracle Fusion Middleware Extensions for Applications provides the FND_DATA_SECURITY PL/SQL package for the data security system.

FUNCTION check_privilege determines whether the user is granted a particular action for a particular row instance, as shown in the following example. The user is determined from the user session context.

This API returns a 1 byte result code:

T - Action is granted.
F - Action is not granted.
E - Error
U - Unexpected error.

PROCEDURE get_security_predicate gets the union of all predicates for the user on an action, as shown in the following example. The user is determined from the user session context.

p_grant_instance_type can take on one of the following values:

INSTANCE - Returns predicate for policies with instance_type = 'INSTANCE' or 'GLOBAL'.
SET - Returns predicate for policies with instance_type = 'SET'.

Note:
'SET' mode does not support aliases.

UNIVERSAL - (Default) Returns predicate for policies with any instance_type.

p_table_alias is appended in front of the column references in the returned x_predicate. It is normally used when two security predicates are going to be ANDed together to use with a select that joins two secured tables. The value passed here should correspond to the table alias that the statement uses for the p_object_name passed to this routine. The default, NULL, means there is no table alias so none is appended.

p_statement_type can take on one of the following values:

OTHER - (Default). The predicate returned is not attached by policy to the base table as is done for VPD. In practice, this allows the predicate to have a sub select against the base table, which allows aliases and may improve performance.

VPD - Pass this type if the predicate is attached by policy to the base table. Use this when VPD uses the returned predicate to control access. In practice, this means the predicate cannot have sub selects against the base table, prevents aliases and may lower performance.

EXISTS - Pass this type if the predicate is simply used to determine if there are any rows at all that are available. The predicate returned is in the format like 'EXISTS...'.

x_return_status is the result of all the operations:
T - Successfully got predicate
E - Error
U - Unexpected error
L - Value too long - predicate too large for database VPD

The return value is all the available predicates from the policies on this action for this user. They are OR'ed together to form a SQL predicate that can be dropped into the WHERE clause to limit rows returned to those that are allowed by the security. Does not include WHERE.

Example 50-30  FUNCTION check_privilege API

FUNCTION check_privilege

p_api_version          IN NUMBER,
/* API Version of this procedure - currently 1.0 (Required) */
p_privilege            IN VARCHAR2,
/* Name of the action (Required) */
p_object_name          IN VARCHAR2,
/* Resource on which the policy should be checked from FND_Objects table (Required) */
p_instance_pk1_value   IN VARCHAR2 DEFAULT NULL,
/* p_instance_pk(1...5)_value (Required) 
Primary key values for the row instance, with order corresponding to the order of the PKs in the FND_Objects table. Most resources have only a few primary key columns so let the higher, unused columns default to NULL. 
NOTE: The caller must pass an actual primary key and it must be the primary key of an actual instance (row). */
p_instance_pk2_value   IN VARCHAR2 DEFAULT NULL,
p_instance_pk3_value   IN VARCHAR2 DEFAULT NULL,
p_instance_pk4_value   IN VARCHAR2 DEFAULT NULL,
p_instance_pk5_value   IN VARCHAR2 DEFAULT NULL
)RETURN VARCHAR2;

Example 50-31  PROCEDURE get_security_predicate

PROCEDURE get_security_predicate
(
    p_api_version          IN NUMBER,
/* API version of this procedure - Currently 1.0 (Required) */
    p_privilege            IN VARCHAR2 DEFAULT NULL,
/* NULL represents all actions so the predicate will not take the action into account. */
    p_object_name          IN VARCHAR2,
/* Object on which the predicate should be checked from FND_Objects table. */
    p_grant_instance_type  IN VARCHAR2 DEFAULT 'UNIVERSAL',
/* SET, INSTANCE (Optional) */
    p_statement)_type      IN VARCHAR2 DEFAULT 'OTHER',
/* (Optional) statement type: 'OTHER', 'VPD', 'EXISTS' = to check existence. */
    x_predicate            OUT NOCOPY VARCHAR2,
    x_return_status        OUT NOCOPY VARCHAR2,
    p_table_alias          IN VARCHAR2 DEFAULT NULL
/* (Optional) */
);
Note:
Data security is not designed as a means to limit the number of rows returned; there must always be another selective WHERE clause before the data security predicate.

Condition predicates must be as fast as possible. All predicates that apply to a particular context must go into the SQL statement that gets executed; therefore, all predicates need to be efficient, whether they reject no rows, a few rows, or many rows. Any sub selects in predicates should be on well-indexed columns. Predicates must execute in linear time, doing simply indexed selects. Predicates that involve connect-by or other network, hierarchical operations are not supported. The suggested approach to hierarchical data representations is building and selecting against a compiled representation of the data.

You should use conditions rather than row instance policies. Row instance policies are policies where each policy maps to exactly one row in the resource table. They generally don’t scale well because they require one policy row for every resource table row. Performance is best when the number of policies that apply in any particular context is low. Condition policies involve just one policy, which specifies an unlimited number of rows. If the normal use case would involve more than a few row instance policies, then you probably need to be change the design to use conditions.

Note:
There are no plans to provide any APIs to answer the question “What users have access to a particular function on a particular row instance?” The reason being is that the condition predicates can reference context that can be driven by the user, like profiles. The only way to answer that question would be to loop through every user and set up their context (including profiles, and so on), and then see if they had access. That is not practical given the large number of users that are possible. For the same reason, data security does not try to answer the question “Does a particular user have access to a particular function?”

50.7 Validating Data Security with Diagnostic Scripts

Oracle Fusion Applications provides WLST (Oracle WebLogic Scripting Tool) scripts written in the Python programming language to help administrators verify that data security setup and configuration definitions are correct for a newly deployed application. Other WLST scripts help administrators verify that applications context setup and configuration definitions are correct for a newly deployed application and that the applications context is created for a logged-in user.

The data security diagnostic scripts may be run in the WLST scripting environment or from the Diagnostic Dashboard application of any Oracle Fusion application. The Diagnostic Dashboard provides administrators with a graphical user interface to execute and monitor diagnostic tests.

For more information about the Diagnostic Dashboard, see the “Running and Working with Diagnostic Tests Manually” section in the Administrator’s Guide.
50.7.1 How to Validate Data Security Configuration with Diagnostic Scripts

The WLST script `datasecurityDiagnostics.py` is provided to help administrators verify that data security setup and configuration definitions are correct for a newly deployed application. Additionally, the script generates a report that system integrators, developers and security managers can further use to diagnose runtime issues related to a logged-in user's ability to access data.

To accomplish these tasks, the script performs these specific functions:

- Validates data security configuration in the `adf-config.xml` file, which is archived in the application EAR file. The script checks the configuration values of the `<sec:JaasSecurityContext>` element to verify that the data security provider is properly configured and outputs the result to the output file `DataSecurityDiagResults.out`.

- Validates that GUID consistency is enabled in the `weblogic-application.xml` file. The script checks the value of the `jps.approle.preserveguid` application parameter and outputs the result to the output file `DataSecurityDiagResults.out`. The application parameter must be set to `TRUE` to support deploying the policy store from a test to a production environment. This ensures the GUID for each application role remain the same when migrated from XML to LDAP or LDAP to LDAP.

- Optionally, takes the running application's session cookie value as an input parameter and, using this session cookie, the script gets the corresponding session object from the data source, inspects the session attributes and outputs the session information to the output file `DataSecurityDiagResults.out`. Using the session cookie, the script gets role information corresponding to the session's logged-in user, along with their access privileges to various database objects and outputs it to the output file `DataSecurityDiagResults.out`.

  Note: If you only want the script to perform the application configuration validation checks, you can skip this step by pressing Enter when prompted to enter the value for session cookie.

Important Note

Before invoking a WLST script you must run the following `wlst.sh` script on Oracle WebLogic Server to ensure that the required JARs are added to the class path.

```bash
>sh $ORACLE_HOME/common/bin/wlst.sh
```

After you invoke the `wlst.sh` script, you can connect to Oracle WebLogic Server in offline mode, that is, the data security script does not require a connection to a running server to operate.

To invoke the data security diagnostic script:

At the offline prompt, enter the following command:

```bash
>ws:/offline> execfile('datasecurityDiagnostics.py')
```

The script prompts you for the following information before outputting the results:

- Output file's directory path where you want the script's output file to be saved.
- Application name for which you want to run the script. The application name is the deployment name in Oracle WebLogic Server. For example, `SalesApp#V3.0`. 
Note that the version part of the application name is specified with a # symbol: for example, #V3.0 in SalesApp#V3.0.

- Application source path of the deployed application. For example, /scratch/myself/view_storage/myself_main_gene_testing/system11.1.4.37.56.69/DefaultDomain/servers/DefaultServer/upload/DemoSecurity/V2.0/app/DemoSecurity.ear. The source path can be obtained from Oracle WebLogic Server Administration Console, under the Deployments section for the application.

- Session cookie value created for the logged-in user for whom you want to validate data security roles and privileges. Optional (you can press Enter to skip). To obtain the session cookie, you must log into the application as the user whose roles and privileges you want to examine. After logging into the application, from the browser, open the cookies window (for example, in Internet Explorer, you can display cookies from the Temporary Internet Files and History Settings dialog). Under the site oracle.com, locate the cookie named <DATABASE_SID>_FND_SESSION and copy the value, which is the required session cookie. If you do not find this named session cookie, it means that the applications context is not created for your application.

Tip:
You can run the application context diagnostic script by pressing Enter when the data security script prompt you to enter the value for session cookie for your application. This will invoke the applications context script and validate the applications context configuration. For details about the applications context script, see How to Validate Applications Context.

### 50.7.2 How to Validate Applications Context

The WLST script applsessionDiagnostics.py is provided to help administrators verify that applications context setup and configuration definitions are correct for a newly deployed application and that the applications context is created for a logged-in user. Additionally, the script generates a report that system integrators, developers and security managers can further use to diagnose the runtime issues related to an Application Session.

To accomplish these tasks, the script performs these specific functions:

- Validates the session filters and filter mapping definitions in the web.xml file, which is archived in the application EAR file. The scripts checks for the presence of the <filter-name>ApplSessionFilter</filter-name> element and verifies that the ApplSessionFilter mapping definition appears immediately after the JpsFilter mapping definition, and then outputs the result to the output file ApplsessionDiagResults.out.

- Connects to the database and gets the Oracle Fusion Applications FND table metadata for the application context. The script checks the value of the Name, Datatype, Precision, and isNullable attributes for each column in the tables, validates the database schema for each table, and then outputs the result to the output file ApplsessionDiagResults.out.

- Optionally, takes the running application’s session cookie value as an input parameter and, using this session cookie, the script gets the corresponding session object from the data source, inspects the session attributes and outputs the applications context session properties to the output file.
ApplsessionDiagResults.out. Using the session cookie, the script gets the applications context properties corresponding to the session's logged-in user, determines whether ApplSession is created properly, and outputs the result to the output file ApplsessionDiagResults.out.

Note: If you only want the script to perform the application configuration validation and the database metadata validation checks, you can skip this step by pressing Enter when prompted to enter the value for session cookie.

- Connects to the database and validates the FUSION.FND_SESSION_MGMT PL/SQL package to check for its consistency. The script checks whether a valid package header and package body is defined for the package and outputs the result to the output file ApplsessionDiagResults.out.

Before you begin:

Before invoking a WLST script you must run the following wlst.sh script on Oracle WebLogic Server to ensure that the required JARs are added to the class path. Use the following command:

>sh $ORACLE_HOME/common/bin/wlst.sh

After you invoke the wlst.sh script, you can connect to Oracle WebLogic Server in offline mode, that is, the data security script does not require a connection to a running server to operate.

To invoke the application context diagnostic script:

At the offline prompt, enter the following command:

>wls:/offline> execfile('applsessionDiagnostics.py')

The script prompts you for the following information before outputting the results:

- Output file's directory path where you want the script's output file to be saved.
- Application name for which you want to run the script. The application name is the deployment name in Oracle WebLogic Server. For example, SalesApp#V3.0. Note that the version part of the application name is specified with a # symbol: for example, #V3.0 in SalesApp#V3.0.
- Application source path of the deployed application. For example, /scratch/ myself/view_storage/myself_main_gene_testing/ system11.1.1.4.37.56.69/DefaultDomain/servers/DefaultServer/ upload/DemoSecurity/V2.0/app/DemoSecurity.ear. The source path can be obtained from Oracle WebLogic Server Administration Console, under the Deployments section for the application.
- Session cookie value created for the logged-in user for whom you want to validate data security roles and privileges. Optional (press Enter to skip). To obtain the session cookie, you must log into the application as the user whose roles and privileges you want to examine. After logging into the application, from the browser, open the cookies window (for example, in Internet Explorer, you can display cookies from the Temporary Internet Files and History Settings dialog). Under the site oracle.com, locate the cookie named <DATABASE_SID>_FND_SESSION and copy the value, which is the required session cookie. If you do not find this named session cookie, it means that the applications context is not created for your application.
50.8 Integrating with Data Security Task Flows

The Oracle Fusion Middleware Extensions for Applications data security task flows are a set of four task flows that provide a simplified user interface for implementing role-based security in Fusion applications. Consider integrating data security task flows into an Oracle Fusion application when your application needs to support an authorized end user's ability to secure business objects in their business domain. For example, these task flows can give Human Resources managers the ability to secure employee records to grant HR representatives access to defined groups of employee records.

Oracle Fusion Middleware Extensions for Applications data security task flows use Oracle Fusion Applications security technology to implement data security policies in Oracle Fusion Applications FND tables and function security policies in the LDAP policy store. At runtime, the task flows implementation performs the necessary backend operations to both secure data exposed by a business object (data security) and to secure the user interface (function security) that displays the business objects.

The following task flows are available and may be integrated using Oracle JDeveloper:

- **Object-centric task flow** lets the end user display and secure a single business object. The flow displays the end user's business object selection and then displays all the application roles that may be granted access to that object. This task flow simplifies the task of securing a business object across the organization.

- **Role-centric task flow** (also called the Profile task flow) lets the end user select an application role profile and secure multiple business objects pertaining to that profile. The flow displays the end user's application role profile selection and displays all the securable business objects that the profile may access. This task flow is also called the profile task flow since every end user is assigned to a profile that corresponds to an enterprise role mapped to a hierarchy of application roles. Note that the role-centric task flow and the object-centric task flow present the end user with different views of the same security functionality. Both enable data security and functionality security policies for permissions that the end user selects across business objects.

- **Instance-level task flow** lets the end user select an instance of a securable business object (one row) for which they have access and then confer access rights to another user or group of users. This task flow provides a way for end users to share access rights with other members of their organization. The scope of this task flow is different from the role-centric and object-centric task flows in that security is limited to the business object instance. The business object itself must already have grants made to the end user who wishes to share access.

- **Role management task flow** lets the end user create and edit application roles. This task flow is particularly useful when the user creates custom business objects and wishes to grant permission to that object for a custom application role.

50.8.1 About Integrating the Data Security Task Flows into Your Application

The process of integrating the data security task flows into an Oracle Fusion application involves understanding the input parameters of the task flow. Your application will use a managed bean to initialize the task flow’s parameters before the application displays the task flow to the end user. The way your application references the values of the input parameters on the bean depends on how you want to display
the task flow. Your application can display the data security task flow one of two ways:

- You can display the task flow in the application's primary browser window.
- You can display the task flow in the application's secondary browser window that displays a new web page and allows the user to view the primary window while working in the task flow.

For example, the object instance task flow user interface is well-suited to run in a dialog. When you run this task flow in a dialog, the user can select an object in the primary browser window, make grants on the selection in the secondary window, and repeat for other objects without needing to reopen the primary window. The other task flows, including the object-centric task flow, profile (role-centric) task flow, and role management task flow, are large enough that you may want to display them in the primary window.

The steps to integrate the data security task flows into your application will depend on the method you choose to display the task flow. However, review the following general steps for an overview of the process.

Before you begin:

Add the data security task flows to your project. Decide whether you want your application to display the task flow in the primary window or in a popup dialog.

To integrate the data security task flows, follow these general steps.

1. Decide whether you want your application to display the task flow in the primary window or in a popup dialog.

2. Create a task flow reference in your application to bind the data security task flow to the ADF Model layer.

   In JDeveloper, the reference will be generated for you when you drag and drop the data security task flow. The way you drag and drop the data security task flow depends on the way your application displays the task flow:

   - How to Configure Data Security Task Flows to Display in the Primary Window
   - How to Configure the Object Instance Task Flow to Display in a Dialog

a. Define the data security task flow’s input parameters so they have page flow scope.

   Page flow scope will allow the values to be passed into the task flow from a managed bean. The ADF Model layer component that you use to define the parameters depends on the way your application displays the task flow:

   - How to Configure Data Security Task Flows to Display in the Primary Window
   - How to Configure the Object Instance Task Flow to Display in a Dialog

b. Create a managed bean and define an initialization method to populate the input parameters for the data security task flows, as shown in Table 50-4.

   The method you define will initialize the values before your application displays the task flow. When displaying the page inside your application’s primary window, the initialization method must also return the value of the flow control outcome you configure in your application’s task flow to invoke the data security task flow. The outcome return value is not needed when displaying a dialog, since the dialog is not invoked the same way.
**c.** Create a navigation button that invokes the initialization method in your method bean.

The ADF Faces component you use to create the button depends on how you want the data security task flow to display:

3. Authorize the data security task flows to make grants in the LDAP policy store. For details see How to Grant the Data Security Task Flows Access to the Application Policy Store.

4. Authorize end user to access the task flow. For details see How to Grant the End User Access to the Data Security Task Flows.

5. Configure the application to access the domain LDAP policy store. For details see How to Map the Application to an Existing Application Stripe.

The following table describes the input parameters that your managed bean must initialize for each task flow.

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Parameters Passed</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object-centric task flow (also referred to as ObjectLevelTF)</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/ObjectLevelTF.xml</td>
<td>Specify the name of the object for which the grant will be managed: objectName Specify the list of role categories from which the available application roles will be fetched: roleCategories Specify the list of securable actions to be displayed in the UI for the object: actions Specify true/false to determine whether the View All column (supports global grants) should appear in the task flow UI: disableViewAll Specify true/false to determine whether the Update All column (supports global grants) should appear in the task flow UI: disableUpdateAll</td>
<td>Create and manage grants to multiple application roles for a single business object.</td>
</tr>
</tbody>
</table>
### Table 50-4  (Cont.) Data Security Task Flows and Their Input Parameters

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Parameters Passed</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role-centric task flow (also referred to as ProfileTF)</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/ProfileTF.xml</td>
<td>Specify the list of role categories from which the available application roles will be fetched: roleCategories</td>
<td>Create and manage grants to a single application role profile for multiple business objects.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specify the list of securable actions to be displayed in the UI for the object: actions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specify true/false to determine whether the View All column (supports global grants) should appear in the task flow UI: disableViewAll</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specify true/false to determine whether the Update All column (supports global grants) should appear in the task flow UI: disableUpdateAll</td>
<td></td>
</tr>
<tr>
<td>Instance-level task flow (also referred to as ObjectInstTF)</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/ObjectInstTF.xml</td>
<td>Specify the name of the parent object from which the object instance will be fetched: objectName</td>
<td>Confer existing grants for a single instance of a business object to another user (or user group).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specify the primary key of the object instance for which grants will be shared: instancePk1 instancePk2 instancePk3 instancePk4 instancePk5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specify the list of securable actions to be displayed in the UI for the object: actions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specify the ID of the customized task flow: taskflowId</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specify the holder of the customized task flow parameters: parameterMap</td>
<td></td>
</tr>
</tbody>
</table>
Table 50-4 (Cont.) Data Security Task Flows and Their Input Parameters

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Parameters Passed</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role management task flow (also referred to as RoleManagementTF)</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/RoleManagementTF.xml</td>
<td>Specify the list of role categories from which the available application roles will be fetched: roleCategories Specify the title of the task flow UI: title</td>
<td>Create and edit custom application roles.</td>
</tr>
</tbody>
</table>

50.8.2 How to Configure Data Security Task Flows to Display in the Primary Window

When you integrate the task flow as a primary window, your application's task flow invokes the data security task flow using a task flow call activity. A control flow case defines the transition (identified with a particular outcome value) between your application's view activity (for the calling web page) and the call activity. A navigation button in the calling web page invokes a method on the managed bean that initializes the task flow's input parameters and returns the expected value of the control flow case outcome. Your application's task flow invokes the data security task flow through the call activity reference that matches the returned outcome.

To integrate a data security task flow with your application so it appears in the primary browser window of the application:

1. In your application's task flow, create a call activity and specify a control flow case from the calling web page's view activity.

   The calling web page is the page in your application where you want the end user to launch the data security UI. This is the page that will be replaced in the browser window when the data security UI is displayed.

2. Drop the desired data security task flow onto the task flow call activity.

3. Edit your application's task flow configuration file to specify the data security task flow's input parameter values on the call activity.

4. Create a managed bean that initializes the data security task flow's input parameters and returns the value of the outcome for the control flow case you specified.

5. Register the managed bean in your application's task flow configuration file.

6. In the web page associated with the view activity, drop an ADF command button and configure the button to invoke the managed bean's initialization method.

50.8.2.1 Create a Task Flow Call Activity in Your Application's Task Flow

You can use a task flow call activity to call any one of the data security task flows from your application's unbounded or bounded task flow. The task flow call activity allows you to call the data security task flows located within the same or a different application. To pass parameters into the data security task flow, you specify input parameter values on the task flow call activity. These values must correspond to the input parameter definitions on the called data security task flow.
The following example shows the task flow call activity definition with a reference to the object-centric data security task flow. The task flow call activity also defines the input parameter values required by the object-centric task flow.

The following example shows the task flow call activity definition with a reference to the role-centric data security task flow. The task flow call activity also defines the input parameter values required by the role-centric task flow.

The following example shows task flow call activity definition with a reference to the role management data security task flow. The task flow call activity also defines the input parameter values required by the role management task flow.

To create the task flow call activity:

1. Open your application's task flow (the calling task flow) in the diagram editor.

2. In the ADF Task Flow page of the Component Palette, drag a Task Flow Call activity and drop it on the calling task flow.

3. In the ADF Task Flow page of the Component Palette, select Control Flow Case and create the control flow case between the source activity (in your calling task flow) and the call activity.

4. In the task flow diagram, enter the outcome value for the control flow case.

   The value of the outcome must match the return value of the task flow initialization method you create in the managed bean. For details about the managed bean, see Initializing the Data Security Task Flow Using a Managed Bean.

5. In the Application Navigator, drag the desired data security task flow and drop it on top of the task flow call activity that is located on the calling task flow.

   This action references the data security task flow as the called task flow in the <task-flow-call> definition. JDeveloper adds the task flow reference to the calling task flow's configuration file.

6. In the editor for the calling task flow, click the Source tab to view the new task flow reference.

7. In the source for the calling task flow, locate the <task-flow-call> element and create the input parameter definitions by copying and pasting from the sample code:

   - If you dropped the object-centric task flow (the activity references task flow <id>ObjectLevelTF</id>), add the input parameter values from Example 50-32.
   - If you dropped the role-centric task flow (the activity references task flow <id>ProfileTF</id>), add the input parameter values from Example 50-33.
   - If you dropped the role management task flow (the activity references task flow <id>RoleManagementTF</id>), add the input parameter values from Example 50-34.

Instead of copying and pasting the input parameter values from the samples, you can also use the Property Inspector to define each input parameter value. However, it is important that the input parameter values you create match the parameter names specified by the called task flow. Copying from the samples ensures the names match exactly.
Example 50-32  ObjectLevelTF Reference in Calling Task Flow Configuration File

```xml
<task-flow-call id="ObjectLevelTF">
  <task-flow-reference>
    <id>ObjectLevelTF</id>
  </task-flow-reference>
  <input-parameter>
    <name>objectName</name>
    <value>${pageFlowScope.objectName}</value>
  </input-parameter>
  <input-parameter>
    <name>roleCategories</name>
    <value>${pageFlowScope.roleCategories}</value>
  </input-parameter>
  <input-parameter>
    <name>actions</name>
    <value>${pageFlowScope.actions}</value>
  </input-parameter>
  <input-parameter>
    <name>disableViewAll</name>
    <value>${pageFlowScope.disableViewAll}</value>
  </input-parameter>
  <input-parameter>
    <name>disableUpdateAll</name>
    <value>${pageFlowScope.disableUpdateAll}</value>
  </input-parameter>
</task-flow-call>
```

Example 50-33  ProfileTF Reference in Calling Task Flow Configuration File

```xml
<task-flow-call id="ProfileTF">
  <task-flow-reference>
    <document>/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/ProfileTF.xml</document>
    <id>ProfileTF</id>
  </task-flow-reference>
  <input-parameter>
    <name>roleCategories</name>
    <value>${pageFlowScope.roleCategories}</value>
  </input-parameter>
  <input-parameter>
    <name>actions</name>
    <value>${pageFlowScope.actions}</value>
  </input-parameter>
  <input-parameter>
    <name>disableViewAll</name>
    <value>${pageFlowScope.disableViewAll}</value>
  </input-parameter>
  <input-parameter>
    <name>disableUpdateAll</name>
    <value>${pageFlowScope.disableUpdateAll}</value>
  </input-parameter>
</task-flow-call>
```

Example 50-34  RoleManagementTF Reference in Calling Task Flow Configuration File

```xml
<task-flow-call id="RoleManagementTF">
  <task-flow-reference>
  </task-flow-reference>
</task-flow-call>
```
50.8.2.2 Initialize the Data Security Task Flow Using a Managed Bean

Managed beans are Java classes that you register with the application in your calling task flow’s configuration file. You will create a method on a managed bean to:

- Initialize the input parameters of the data security task flows.
- Return a value that matches the outcome of the calling task flow.

When your application runs, and the end user clicks the button in the calling web page, the method is invoked and the properties are declared, allowing the called data security task flow input parameters to be populated with the declared values.

The following example shows the additional source code that your managed bean must include to initialize the object-centric (ObjectLevelTF) task flow.

The following example shows the additional source code that your managed bean must include to initialize the role-centric (ProfileTF) task flow.

The following example shows the additional source code that your managed bean must include to initialize the role management (RoleManagementTF) task flow.

To populate task flow input parameters with a managed bean:

1. Use the New Gallery to create the managed bean class and name it for the called task flow.

   For example, you might create a bean named DSRoleManage.java for the RoleManagementTF task flow.

2. In the class editor, create a method that you want to use to initialize the task flow input parameters.

   For example, you might create a method for the RoleManagementTF task flow, initializeRoleManagement().

3. In the method definition, create the input parameter property definitions by copying and pasting from the sample code into the method definition:

   - If you want to initialize the object-centric task flow, add the input parameter property declarations from Example 50-35.

   - If you want to initialize the role-centric task flow (also called the Profile task flow), add the input parameter property declarations from Example 50-36.

   - If you want to initialize the role management task flow, add the input parameter property declarations from Example 50-37.
Copying from the samples ensures the input parameters names match those defined by the data security task flow.

4. In the source you pasted, edit the input parameter property definitions to specify the default values for your application.

5. Enter a return value for the initialization method that matches the outcome of the control flow case you specified in your calling task flow.

For example, for the outcome value "start", your method should show:

```
return "start";
```

For details about creating the control flow case in your application’s task flow, see Creating a Task Flow Call Activity in Your Application’s Task Flow.

**Example 50-35  Source for Populating the ObjectLevelTF Input Parameters**

```
//Source for managed bean method to populate and pass ObjectLevelTF parameters
String objectName = "TEST_DS_EMP";
List actions = new ArrayList<ActionMap>();

//for action 1
List<String> instanceSets1 = new ArrayList<String>();
instanceSets1.add("TEST_EMP_IS1");
instanceSets1.add("TEST_EMP_IS2");
instanceSets1.add("TEST_EMP_IS3");

ActionMap action1 = ActionMap.getActionMapForObjectUI("View", "VIEW_M",
    "ViewPermSet", false, instanceSets1);
actions.add(action1);

//for action 2
List<String> instanceSets2 = new ArrayList<String>();
instanceSets2.add("TEST_EMP_IS2");
instanceSets2.add("TEST_EMP_IS3");
instanceSets2.add("TEST_EMP_IS4");

ActionMap action2 = ActionMap.getActionMapForObjectUI("Update", "UPDATE_M",
    "UpdatePermSet", false, instanceSets2);
actions.add(action2);

//populate role categories
List roleCategories = new ArrayList<String>();
roleCategories.add("Category1");
roleCategories.add("Category2");
roleCategories.add("Category3");

//set the parameters in pageflow scope
AdfFacesContext context = AdfFacesContext.getCurrentInstance();
Map pageFlowScope = context.getPageFlowScope();
pageFlowScope.put("objectName", objectName);
pageFlowScope.put("roleCategories", roleCategories);
pageFlowScope.put("actions", actions);
```

**Example 50-36  Source for Populating the ProfileTF Input Parameters**

```
//Source for managed bean method to populate and pass ProfileTF parameters
List roleCategories = new ArrayList<String>();
roleCategories.add("Category1");
```
roleCategories.add("Category2");

List actions = new ArrayList<ActionMap>();

// for action 1
List<ProfileActionObject> profileObjects1 = new ArrayList<ProfileActionObject>();
profileObjects1.add(ProfileActionObject.getProfileActionObject("View", "TEST_DS_EMP", "VIEW_EMP_M","ViewEmpPermSet"));
profileObjects1.add(ProfileActionObject.getProfileActionObject("View", "TEST_DS_EMP1", "VIEW_EMP1_M","ViewEmp1PermSet"));
profileObjects1.add(ProfileActionObject.getProfileActionObject("View", "TEST_DS_EMP2", "VIEW_EMP2_M","ViewEmp2PermSet"));
ActionMap action1 = ActionMap.getActionMapForProfileUI("View", profileObjects1);
actions.add(action1);

// for action 2
List<ProfileActionObject> profileObjects2 = new ArrayList<ProfileActionObject>();
profileObjects2.add(ProfileActionObject.getProfileActionObject("Update", "TEST_DS_EMP1", "UPDATE_EMP1_M","UpdateEmp1PermSet"));
profileObjects2.add(ProfileActionObject.getProfileActionObject("Update", "TEST_DS_EMP2", "UPDATE_EMP2_M","UpdateEmp2PermSet"));
ActionMap action2 = ActionMap.getActionMapForProfileUI("Update", profileObjects2);
actions.add(action2);

AdfFacesContext context = AdfFacesContext.getCurrentInstance();
Map pageFlowScope = context.getPageFlowScope();
pageFlowScope.clear();
pageFlowScope.put("roleCategories", roleCategories);
pageFlowScope.put("actions", actions);
pageFlowScope.put("disableViewAll", false);
pageFlowScope.put("disableUpdateAll", false);

Example 50-37 Source for Populating the RoleManagementTF Input Parameters

// Source for managed bean method to populate & pass RoleManagementTF parameters
AdfFacesContext context = AdfFacesContext.getCurrentInstance();
Map pageFlowScope = context.getPageFlowScope();
pageFlowScope.clear();
List roleCategories = new ArrayList<String>();
roleCategories.add("Category1");
roleCategories.add("Category2");
roleCategories.add("Category3");
pageFlowScope.put("roleCategories", roleCategories);
pageFlowScope.put("title", "Role Management");

50.8.2.3 Register the Managed Bean with Your Application's Task Flow

To declare the manage bean in the calling task flow's configuration file, you must enter a bean identifier name, the class path for the bean, and you must specify backing bean scope. The following example uses the bean identifier roleManageBean to match the Action attribute definition specified on the button component shown in Example 50-39.

To register the managed bean with the calling task flow:

1. Open your application's task flow (the calling task flow) in the diagram editor.
2. In the editor for the task flow, click the Source tab.
3. In the source for the calling task flow, declare the managed bean by creating the `<managed-bean>` element similar to the sample shown in the previous example.

**Example 50-38 Managed Bean Declaration in Calling Task Flow Configuration File**

```xml
<adfc-config xmlns="http://xmlns.oracle.com/adf/controller" version="1.2">
  <task-flow-definition id="MyCallingTaskFlow">
    <default-activity id="__1">CallRoleManageTF</default-activity>
    <managed-bean id="__4">
      <managed-bean-name id="__5">roleManageBean</managed-bean-name>
      <managed-bean-class id="__2">oracle.apps.fnd.applcore.dataSecurity.view.RoleManage</managed-bean-class>
      <managed-bean-scope id="__3">backingBean</managed-bean-scope>
    </managed-bean>
  </task-flow-definition>
  ...
</adfc-config>
```

50.8.2.4 Creating a Command Button to Invoke the Task Flow in the Primary Window

Create an ADF command button component in the web page associated with the source activity of the control flow case you create in the calling task flow. When the end user clicks the button, the button will cause the data security task flow to display. You will need to configure the Action attribute of the button so a click by the end user invokes the initialization method of your managed bean.

The following example shows a command button component with an Action attribute that invokes the method `initializeRoleManagement()` on the bean referenced by the bean identifier `roleManageBean`. The name of the identifier corresponds to the managed bean declaration you create when you edit the calling task flow configuration file, as described in Registering the Managed Bean with Your Application's Task Flow.

**Example 50-39 Button Component with Action Attribute to Invoke Bean Method**

```xml
<af:commandButton text="Manage Roles" id="cb1"
  action="#{backingBeanScope.roleManageBean.initializeRoleManagement}"/>
```

50.8.3 How to Configure the Object Instance Task Flow to Display in a Dialog

When you integrate a data security task flow as a dialog, your application’s task flow invokes the task flow using an executable in the ADF Model layer. This executable is defined by JDeveloper when you drop the data security task flow as region onto a dialog component that you add to your application’s calling web page. The web page the defines the region is associated with a view activity in your application task flow; no task flow control flow case is needed to invoke the data security task flow.

A popup button in the calling web page defines a listener that invokes a method on the managed bean to initialize the task flow’s input parameters. The button then displays the dialog with a region that invokes the task flow using the executable defined on the calling page’s definition. When the user clicks the dialog close button, a listener (for the dialog’s close button) saves the end user’s sections from data security UI.

To integrate the object instance task flow (ObjectInstTF) with your application so it appears in a dialog (as a secondary browser window):

1. In your application’s task flow, double-click the view activity associated with the web page that end users will use to open the dialog.
2. In the web page, drop an ADF command button component.

3. Drop an ADF popup component onto the button and configure a popup fetch listener to invoke an initialization method on a managed bean.

4. Drop an ADF dialog inside the popup component and configure a dialog listener to invoke a method to save grants made by the user.

5. Drop the object instance task flow as a region onto the dialog component.

6. Edit the page definition file for the calling page to specify the data security task flow’s input parameter values on the task flow executable.

7. Create a managed bean and define a method to initialize the task flow parameters before the dialog displays.

8. Define another method on the managed bean to trigger the task flow save action and save the grants into the database.

9. Register the managed bean in your application's task flow configuration file.

50.8.3.1 Create the Task Flow Executable in the Region Page Definition File

When you drop a data security task flow onto a web page to create an ADF region, JDeveloper adds an af:region tag to the page. The af:region tag references an object that implements RegionModel, as shown in #unique_1712/unique_1712_Connect_42_BABGFDE. JDeveloper also adds a task flow binding to the <executables> element of the page definition file for the page that defines the ADF region. The task flow binding provides a bridge between the ADF region and the data security task flow. It binds a specific instance of an ADF region to the data security task flow and maintains all information specific to the task flow. The taskFlowId attribute specifies the directory path and the name of the source file for the bounded task flow. To pass parameters into the data security task flow, you must specify input parameter values on the task flow binding. These values must correspond to the input parameter definitions on the called data security task flow.

The following example shows task flow binding in the page definition file with a reference to the object-instance data security task flow. The task flow binding also defines the input parameter values required by the object-instance task flow.

To create the task flow binding definition:

1. Open your application's task flow (the calling task flow) in the diagram editor.

2. In the ADF Task Flow page of the Component Palette, drag a View activity and drop it on the calling task flow.

3. In the ADF Task Flow page of the Component Palette, select Control Flow Case and create the control flow case between the source activity (in your calling task flow) and the view activity.

4. In the task flow diagram, enter the outcome value.

   The value of the outcome should match the return value of the initialization method you create in the managed bean used to populate the data security input parameters. For details about the managed bean, see Initializing the Data Security Task Flow Using a Managed Bean.

5. In the Application Navigator, drag the object instance task flow and drop it on top of the task flow call activity that is located on the calling task flow.
This action defines the data security task flow as the called task flow using a `<task-flow-call>` definition. The task flow call definition appears in the calling task flow configuration file.

6. In the editor for the task flow, click the **Source** tab to view the new task flow reference.

7. In the source for the calling task flow, locate the `<task-flow-call>` element and create the input parameter definitions by copying and pasting from the sample code:

   - If you dropped the object-centric task flow (activity will show task flow reference `<id>ObjectLevelTF</id>`), add the input parameter values from Example 50-32.
   - If you dropped the role-centric task flow (activity will show task flow reference `<id>ProfileTK</id>`), add the input parameter values from Example 50-33.
   - If you dropped the role management task flow (activity will show task flow reference `<id>RoleManagementTF</id>`), add the input parameter values from Example 50-34.

Instead of copying and pasting the input parameter definitions from the samples, you could also use the Property Inspector to define each input parameter. However, it is important that the input parameter definitions you create match the parameter names specified by the called task flow. Copying from the samples ensures the names match exactly.

**Example 50-40  **ObjectInstTF Call Entry in Page Definition File**

```
<taskFlow id="ObjectInstTF1"
  taskFlowId="/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/
   taskflow/ObjectInstTF.xml#ObjectInstTF"
  Refresh="ifNeeded"
  xmlns="http://xmlns.oracle.com/adf/controller/binding">
  <parameters>
    <parameter id="objectName" value="#{pageFlowScope.objectName}" xmlns="http://xmlns.oracle.com/adfm/uimodel" />
    <parameter id="actions" value="#{pageFlowScope.actions}" xmlns="http://xmlns.oracle.com/adfm/uimodel" />
    <parameter id="instancePK1" value="#{pageFlowScope.instancePK1}" xmlns="http://xmlns.oracle.com/adfm/uimodel" />
    <parameter id="instancePK2" value="#{pageFlowScope.instancePK2}" xmlns="http://xmlns.oracle.com/adfm/uimodel" />
    <parameter id="instancePK3" value="#{pageFlowScope.instancePK3}" xmlns="http://xmlns.oracle.com/adfm/uimodel" />
    <parameter id="instancePK4" value="#{pageFlowScope.instancePK4}" xmlns="http://xmlns.oracle.com/adfm/uimodel" />
    <parameter id="instancePK5" value="#{pageFlowScope.instancePK5}" xmlns="http://xmlns.oracle.com/adfm/uimodel" />
    <parameter id="taskflowId" value="#{pageFlowScope.taskflowId}" xmlns="http://xmlns.oracle.com/adfm/uimodel" />
    <parameter id="parameterMap" value="#{pageFlowScope.parameterMap}" xmlns="http://xmlns.oracle.com/adfm/uimodel" />
  </parameters>
</taskFlow>
```

### 50.8.3.2 Creating a Command Button to Invoke the Popup Dialog

Your web page should display a button that will invoke the popup operation.
Create an ADF command button component in the web page associated with the view activity of the calling task flow. Then drop an ADF popup component in the panel that contains the button. Then drop an ADF `showPopupBehavior` operation onto the button and set it to the ID of the popup component. Finally, drag a dialog onto the popup component.

You will also need to specify a listener for both the popup component and the dialog component. These listeners invoke methods on the managed bean to initialize the input parameters and save the user selections.

The following example shows a command button component with the `showPopupBehavior` operation nested on the button. The popup component appears in the same panel as the button and defines the `popupFetchListener` property to identify the `launchPolicy()` method on the bean referenced by the bean identifier `objectInstanceBean`. The dialog component defines the `dialogListener` property to identify the `okCreatePolicy()` method on the bean referenced by the same bean identifier. The name of the identifier corresponds to the managed bean declaration you create when you edit the calling task flow configuration file.

Note that the ADF region component element shown in the example will be created when you drop the data security object instance task flow as a region.

**Button Component with Show Popup Operation to Invoke Dialog**

```html
<af:panelHeader id="phl12" text="Test Instance UI">
  <af:panelGroupLayout id="pgl13" layout="horizontal">
    <af:commandButton id="cb6" text="Launch Object Instance Popup"
      partialSubmit="true">
      <af:showPopupBehavior popupId="objInst"/>
    </af:commandButton>
    <af:popup id="objInst" contentDelivery="lazyUncached"
      popupFetchListener="#{backingBeanScope.objectInstanceBean.launchPolicy}"
      childCreation="deferred">
      <af:dialog id="polDiag" modal="true" title="Object Instance UI"
        affirmativeTextAndAccessKey="Save and Close" resize="on"
        contentWidth="600" contentHeight="330"
        stretchChildren="first"
        dialogListener="#{backingBeanScope.objectInstanceBean.okCreatePolicy}"
      >
        <af:region value="#{bindings.ObjectInstTF1.regionModel}" id="r1"/>
      </af:dialog>
    </af:popup>
  </af:panelGroupLayout>
</af:panelHeader>
```

### 50.8.3.3 Initialize the Object-Instance Task Flow Using a Managed Bean

Managed beans are Java classes that you register with the application in your calling task flow’s configuration file. You will define two methods on a managed bean:

- A popup fetch listener method to add properties to the managed bean that will initialize the input parameters of the data security task flows.
- A dialog listener method to save the grants made by the end user to the database when the user closes the dialog.

When your application runs, and the end user clicks the button in the calling web page, the popup fetch listener method is invoked and the properties are declared, allowing the called data security task flow input parameters to be populated with the declared values.
The following example shows additional source code that your managed bean must include to initialize the object-instance (ObjectInstTF) task flow.

The following example shows the source code you must add for the method your dialog listener invokes. In this case, the method is named `okCreatePolicy()` to match the method invoked by the dialog listener in `#unique_1712/unique_1712_Connect_42_BABGFDGE`.

To populate task flow input parameters with a managed bean:

1. Use the New Gallery to create the managed bean class and name it for the called task flow.

   For example, you might create a bean named `DSObjectInstance.java` for the object instance task flow.

2. In the class editor, create a method that you want the popup listener to invoke to initialize the task flow input parameters.

   For example, you might create a method `launchPolicy()` named for the method invoked by the popup listener (shown in `#unique_1712/unique_1712_Connect_42_BABGFDGE`).

3. In the method definition, create the input parameter property definitions by copying and pasting from Example 50-41 into the new initialization method definition.

   Copying from the sample ensures the input parameters names match those defined by the object instance task flow.

4. In the source you pasted, edit the input parameter property definitions to specify the default values for your application.

5. In the class editor, create a method that you want the dialog listener to invoke to save the grants after the user closes the dialog.

   For example, you might create a method `okCreatePolicies()` named for the method invoked by the dialog listener (shown in `#unique_1712/unique_1712_Connect_42_BABGFDGE`).

6. In the method definition, create the save operation by copying and pasting from Example 50-42 into the new save grants method definition.

   Copying from the sample ensures the method `saveInstanceGrants()` is called exactly as shown.

**Example 50-41 Source for Populating the ObjectInstTF Input Parameters**

```java
//Source for managed bean to populate and pass ObjectInstTF parameters
public void launchPolicy(PopupFetchEvent popupFetchEvent) {
    List actions = new ArrayList<ActionMap>();

    ActionMap action1 = ActionMap.getActionMapForInstanceUI("View", "VIEW_EMP_M");
    actions.add(action1);

    ActionMap action2 = ActionMap.getActionMapForInstanceUI("Update", "UPDATE_EMP_M");
    actions.add(action2);

    ActionMap action3 = ActionMap.getActionMapForInstanceUI("Create", "CREATE_EMP_M");
}
```

Implement Oracle Fusion Data Security 50-51
actions.add(action3);
ActionMap action4 =
ActionMap.getActionMapForInstanceUI("Delete", "DELETE_EMP_M");
actions.add(action4);
ActionMap action5 =
ActionMap.getActionMapForInstanceUI("CustomFSOnlyAction",
"CUSTOMFSONLYACTION_EMP_M");
actions.add(action5);

AdfFacesContext context = AdfFacesContext.getCurrentInstance();
Map pageFlowScope = context.getPageFlowScope();
pageFlowScope.clear();
pageFlowScope.put("actions", actions);
pageFlowScope.put("objectName", "TEST_DS_EMP");
pageFlowScope.put("instancePK1", "2");
pageFlowScope.put("instancePK2", null);
pageFlowScope.put("instancePK3", null);
pageFlowScope.put("instancePK4", null);
pageFlowScope.put("instancePK5", null);

// If you want to use a customized task flow instead of people picker to select
// users, you need specify values for the taskflowId and parameterMap.
// Map<String, Object> paraMap = new HashMap<String, Object>();
// paraMap.put("pageTitle", "Custom User Selection");
// pageFlowScope.put("parameterMap", paraMap);
// Taskflow name
// pageFlowScope.put("taskflowId", "/WEB-INF/CustomDSUserTF.xml#CustomDSUserTF");

Example 50-42  Source for Saving the ObjectInstTF Grants

//Source for managed bean to save end user grant
public void okCreatePolicy(DialogEvent dialogEvent)
{
   DataSecurityUIUtils.saveInstanceGrants(dialogEvent);
   //Add your custom code if needed
}

50.8.3.4 Register the Managed Bean with Your Application's Task Flow

To declare the managed bean in the calling task flow's configuration file, you must enter
a bean identifier name, the class path for the bean, and you must specify backing bean
scope. The following example uses the bean identifier objectInstanceBean to
match the bean references specified in the listener properties shown in #unique_1712/
unique_1712_Connect_42_BABGFDE.

To register the managed bean with the calling task flow:

1. Open your application's task flow (the calling task flow) in the diagram editor.
2. In the editor for the task flow, click the Source tab.
3. In the source for the calling task flow, declare the managed bean by creating the
   <managed-bean> element similar to the sample shown in the previous example.

Example 50-43  Managed Bean Definition in Calling Task Flow Configuration File

<adf-config xmlns="http://xmlns.oracle.com/adf/controller" version="1.2">  
<task-flow-definition id="MyCallingTaskFlow">
50.8.4 How to Grant the Data Security Task Flows Access to the Application Policy Store

A grant must be added to the jazn-data.xml policy store to allow your application to provision the LDAP policy store. The LDAP policy store is secured so that only authorized applications can make API calls needed to create and update grants in the store. For this purpose, a code source grant must be made to authorize the implementation code of the data security task flows to make credential store and policy store API calls. The following example shows the code source grant where <application-name> is the application name specified in the jazn-data.xml policy store.

To enable the application to access and update the domain policy store:

1. In JDeveloper, open the jazn-data.xml file in the overview editor.
2. In the source for the jazn-data.xml file, add the code source grant shown in the previous example to the policy store and enter the name of your application in the <url> and <name> definitions.
   
   The application name you enter must match the application name identified in the policy store definition.

Example 50-44 Grant to Enable Policy Store Provisioning by Data Security Task Flow Source Code

50.8.5 How to Grant the End User Access to the Data Security Task Flows

The data security task flows, when integrated into your application, behave like other web application resources secured by ADF Security. By default, ADF Security locks down application resources and therefore requires that you grant access rights to the members of the application roles for the task flows.
To grant view access to the task flows, you define a OPSS permission grant defined by the `oracle.adf.controller.security.TaskFlowPermission` class. The following example shows the permission definition that grants the users (identified by `<grantee>` in your application) view access rights to the task flows.

Note that the resources in the permission grant are identified by regular expression metacharacters `.*` (dot followed by an asterisk). This expression denote any number of arbitrary characters and effectively grants view rights on all task flows in the Oracle Fusion Applications data security path `/WEB-INF/oracle/apps/fnd/appcore/dataSecurity/ui/taskflow/`.

The grantee of the permission are the application roles that your application specifies. If you are using custom application roles not defined by Oracle Fusion Applications, a security manager will need to configure these application roles using Oracle Authorization Policy Manager. For example, an application role requires a role category definition.

To enable function security for the data security task flows:

1. In JDeveloper, open the `jazn-data.xml` file in the overview editor.
2. In the source for the `jazn-data.xml` file, add the permission definition shown in the previous example to the policy store and define the grantee.

   Grantee are typically application roles that your application defines. A grant is always made to a single grantee. When you need to grant view permission to more than one grantee, create duplicate grants and name the grantee in each.


Example 50-45  Grant to View Data Security Task Flows

```xml
<grant>
  <grantee>
    ...
  </grantee>
  <permissions>
    <permission>
      <class>oracle.adf.controller.security.TaskFlowPermission</class>
      <name>/WEB-INF/oracle/apps/fnd/appcore/dataSecurity/ui/taskflow/.*</name>
      <actions>view</actions>
    </permission>
  </permissions>
</grant>
```

50.8.6 How to Map the Application to an Existing Application Stripe

Before you deploy the application, you need to identify the application stripe in the production environment. When deployed, the application will make use of the application roles and security policies defined by the application stripe. The following example shows the `web.xml` entry to identify the existing application stripe.

To map the application to the deployed application stripe:

1. In JDeveloper, open the `web.xml` file in the overview editor.
2. In the source for the `web.xml` file, add the web application initialization parameter shown in the previous example beneath the JPS filter.

   The application name you enter must match the application name identified in the policy store definition.

**Example 50-46  `web.xml` Parameter Identifies Deployed Application Stripe**

```xml
<init-param>
    <param-name>application.name</param-name>
    <param-value><application-name></param-value>
</init-param>
```
This chapter describes how to implement application user sessions in an Oracle Fusion application to allow applications to store security and application context on the user session.

This chapter includes the following sections:

- Introduction to Application User Sessions
- Configuring Your Project to Use Application User Sessions
- Accessing Properties of the Applications Context

### 51.1 Introduction to Application User Sessions

Configuring your user interface project for an application user session is a requirement whenever you want to secure data and interact with Oracle Fusion Data Security. Additionally, before you can run and test your application from a task flow or web page, you should configure your user interface project to use an application user session.

The application user session is used to store user and application context from the time the user logs in until log out. When the application user session is implemented, the Oracle Fusion application can easily reconnect to the same user session for each request, maintaining the user context over the duration of the user's session without the overhead of having to obtain and initiate a database connection each time. The actual connection used is not guaranteed to be the same between requests. Application user session roles can be enabled for a user, which can dictate what data is available to the user based strictly on existing data security policies.

The application user session stores common information used in Oracle Fusion Applications as session attributes and includes basic information about user identity and language preferences, as well as context important to particular applications. Specifically, session information includes the session ID, current user information, current language, date and number formatting, and other similar properties. Session attributes can also be used to track application specific information such as, the current user's shopping cart, the country selection, or the currently selected operating unit.

Application user session namespaces are where attributes on the session are stored. These attributes are then available over multiple requests whenever the session is attached.

Oracle Fusion Applications maintains its own namespaces - one for tracking security information, and another that developers can use to store attributes that they need to track over the life of a session.

Additionally, developers can create their own namespaces for any product specific attributes that they need to track over the life of a session. For example, when a large
number of attributes exists, developers may want to create their own namespaces to group the attributes together more cleanly.

Oracle Fusion Middleware Extensions for Applications provides covers on top of the routines for getting attributes. To access the attributes of the application context, APIs exist in both PL/SQL and Java, as described in Accessing Properties of the Applications Context.

51.2 Configuring Your Project to Use Application User Sessions

When you create a user interface project to test or run a task flow (anything that contains a .jspx file) you need to enable application user sessions for any JSPX pages or task flows that you have created in your user interface project.

If the user interface project provides task flows that are only called from a page in another project, then there is no need to configure your project to use sessions.

51.2.1 How to Configure Your Project to Use Application User Sessions

By default, application user sessions are not enabled for your project. If you wish to access this functionality, you must configure your project.

Before you begin:

It may be helpful to have an understanding of application user sessions. For more information, see Implementing Application User Sessions.

You may need to complete these tasks:

- If you have any web services projects, you must configure them to maintain the application user session across the web service request. For more information, see Maintaining Application Session Context Across Web Service Requests.

- If you have any portlet projects, you must configure them to maintain the application user session across the web service request. For more information, see Maintaining Application Session Context Across Web Service Requests.

To configure your project to use application user sessions:

1. In your Application Navigator, select your data model project and then right-click and choose Project Properties. In the Categories tree, select Libraries and Classpath and verify that the Applications Core and Web Services Data Control libraries have been added.

2. In your Application Navigator, select the web.xml file in the WEB-INF folder of your user interface project. Double-click to open the file. In the Categories tree, select Filters to create a new filter. Enter the following information:

   Filter Name: Enter ApplSessionFilter

   Filter Class: Enter oracle.apps.fnd.applcore.common.ApplSessionFilter

3. Select the Source view to manually modify the web.xml source to add the ApplSessionFilter mapping definition into the same section where other filters are defined—immediately after the JpsFilter mapping definition and before any other definitions.

   The following example shows the ApplSessionFilter mapping definition to add.
It is important that you add this filter mapping immediately after the JpsFilter mapping definition and before any other definitions. This is to ensure that the ApplSessionFilter servlet filter executes immediately after the JpsFilter servlet handles authentication. Normally, this does not make a difference, however there are cases, such as customization code, where this is required.

You can create the above through the Filter Mappings tab (with Mapping Type set to Servlet, and Mapping set to Faces Servlet, and Dispatcher Type set to Forward, Request) but to change the ordering of the filter mapping, you must modify the web.xml file directly.

4. Save all changes.

**Example 51-1 Creating a New Filter Mapping Definition**

```xml
<filter-mapping>
  <filter-name>ApplSessionFilter</filter-name>
  <servlet-name>Faces Servlet</servlet-name>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

You should also stop and restart any server processes that you have running to make sure JDeveloper notices this new change. At this point you can run your page with application user sessions enabled, but you will always be running as the anonymous user. If you wish to require that users to authenticate, you will need to enable authentication and define some users and roles, as described in Adding Function Security to the Application.

### 51.2.2 How to Configure the ADF Business Component Browser

The steps in How to Configure Your Project to Use Application User Sessions can be used to configure the ADF Business Component Browser to run in a mode that supports application user sessions.

If you are running a standalone Java program or a JUnit test, you must explicitly call the ApplSessionUtil.initializeSession API at the beginning of your program to create an applications context object. For more information about how to call the ApplSession.initializeSession API, see Initializing Sessions.

### 51.2.3 How to Use the ApplSession Logger for Troubleshooting

One of the most useful diagnostic tools for applications configured for application user sessions is the ApplSession logger. The specific logger for application users sessions logging is named oracle.apps.fnd.appcore.common.session (also called the ApplSession logger). You use this logger to capture runtime traces messages that are specific to application user sessions. By default, the logger records messages at the WARNING level. If you need to log additional details as part of troubleshooting, you can increase the ApplSession log level to adjust the amount of information that is recorded. For example, more detailed information may help you to quickly identify the origin of an application user sessions error.

In JDeveloper, when you configure a logger and run the application, JDeveloper displays runtime messages recorded by the logger in the Log window and a system log file. You configure logging in JDeveloper using the editor for Oracle Diagnostic Logging Configuration. After you have created a log, you can view and filter the log
messages with Oracle Diagnostic Log Analyzer. This tool allows you to set filters for
different log levels, define message time frames, and search on message text.

After the application is deployed to standalone Oracle WebLogic Server, a system
administrator uses Oracle Enterprise Manager Fusion Applications Control to adjust
the log level and to view the logger’s recorded messages. The system administrator
may increase the ApplSession log level to gather detailed information for a particular
managed server.

For more information about adjusting log levels for the ApplSession logger, using
Fusion Applications Control, see the "Configuring ApplSession Log Levels" section in
the Administrator’s Guide.

Related Links
The following documents provide additional information about Oracle Fusion
Middleware topics discussed in this section:

• For more information about the JDeveloper log tools, see the ”Testing and
  Debugging ADF Components” chapter of the Developing Fusion Web Applications
with Oracle Application Development Framework, in the Oracle Fusion Middleware
Online Documentation Library.

• For more information about Oracle Fusion Middleware logging functionality, see
  the ”Managing Log Files and Diagnostic Data” chapter of the Administering Oracle
  Fusion Middleware, in the Oracle Fusion Middleware Online Documentation
  Library.

51.2.4 What Happens at Runtime: How the Application User Session is Used

When you run your page, the OPSS layer checks if authentication is required. When
access to the page requires authentication of the user, an authentication dialog
prompts the user to log in. After the user completes login, the ApplSessionFilter
is triggered and establishes an applications user session for the user based on the
Subject asserted by the OPSS layer. For example, if you login as OPERATIONS/
welcome, an application user session for the OPERATIONS user is created and set as
the current session. Your page can access this session through the ApplSessionUtil
class, as described in Accessing Properties of the Applications Context.

For details about enforcing security and granting access to application resources while
testing the applications, see Adding Function Security to the Application.

51.3 Accessing Properties of the Applications Context

The applications context is a set of properties relevant to applications that is stored on
the application user session as a series of name-value pairs. You can access the core
application security context in one of two ways:

• Through the ApplSession.java class in the
  oracle.apps.fnd.applicore.common package

• Through the FND_GLOBAL package in PLSQL

The list of context attributes includes information such as current user name and the
current language. The core attributes that are now supported were derived from the
following:

• Customization layer hierarchy.
The following is the list of context attributes that are automatically captured and maintained in the ApplSession context. The values listed are the exact names of the attributes as they are defined in the session context. Note that developers can add their own custom attributes as well.

**Security and Customization attributes:**

- **USER_GUID** - The unique GUID that identifies the currently logged in user.
- **USER_NAME** - The name of the currently logged in user.
- **PRODUCT_FAMILY** - The current active product family.
- **PRODUCT** - The current active product.
- **INDUSTRY** - The current active industry.
- **TERRITORY** - The current active territory.
- **SITE** - Returns the constant value SITE.
- **GLOBAL** - Returns the constant value GLOBAL.
- **ADDTL_CUSTOM_LEVEL** - Additional customization level is a context property that can be customized by developers.
- **INDUSTRY_IN_TERRITORY** - The current active industry in a particular territory.
- **ROLES** - A list of roles (both enterprise roles and OPSS application roles) that are currently active. (Assigned to the currently logged in user). If the application session changes, then the roles associated with the new application will be appended onto the session list.

**Language attributes:**

- **LANGUAGE** - The language tag representing the current language.
- **NLS_LANG** - The two-letter database language code. Derived from the language.
- **NLS_LANGUAGE** - The database language. Derived from the language.
- **NLS_SORT** - String sorting logic in database. This is from linguistic sorting support project.
- **TERRITORY** - Listed above as part of the customization context.
- **DATE_FORMAT** - Format mask pattern for date parsing and formatting.
- **TIME_FORMAT** - Format mask pattern for time parsing and formatting. This includes time zone formatting.
- **GROUPING_SEPARATOR** - Grouping separator for number formatting.
- **DECIMAL_SEPARATOR** - Decimal separator for number formatting.
- **TIME_ZONE** - User’s preferred time zone in Oracle E-Business Suite (EBS) R12.
- **CURRENCY** - The current currency code.
• **CLIENT_ENCODING** - Client native encoding used for file uploading, downloading, export, and attachment.

**Note:**
All language context attributes are handled using Java conventions, except for those that are explicitly prefixed with NLS. For example, `getLanguage()` returns en-US (corresponding to "AMERICAN" in the database) and `getDateFormat()` returns dd-MMM-yy (corresponding to DD-MON-RR in the database).

**Miscellaneous attributes:**
- **TRACE_LEVEL** - The current tracing level when tracing is turned on.
- **MODULE** - Stores the current module for tracing purposes.
- **ACTION** - Stores the current action, such as page, being taken for tracing.
- **ACCESSIBILITY_MODE** - The current accessibility mode.

The stored name-value pairs are partitioned into separate namespaces. Oracle Fusion Applications creates namespaces to store the context attributes.

**Note:**
The actual names of these namespaces and which attributes are used in which namespace is an implementation detail that you do not need to be aware of.

Developers can access the attribute-storage namespaces through the standard APIs that are detailed below.

Developers may also choose to define their own namespaces, especially if they have several attributes they wish to store on the session. Developers may currently choose among the following APIs for initializing namespaces:

- **Java:** `ApplSession.initializeNamespace(String namespaceName)`
- **PL/SQL:** `fnd_global.initialize_namespace (namespace_name IN VARCHAR2);`

The Java and PL/SQL initializeNamespace routines are identical, just invoked from different layers—these will dynamically create a new namespace associated with the currently attached session, which you can then access and retrieve session attributes from for the duration of that session.

### 51.3.1 How to Access Sessions Using Java APIs

In Java, the applications context is accessed through the `ApplSession.java` and `ApplSessionUtil.java` classes, which can be found in the `oracle.apps.fndapplcore.common` package. Each of the attributes listed above have corresponding APIs in the `ApplSession` class, along with a corresponding static API in the `ApplSessionUtil` class for easier access.

For more information, see the javadoc included with Oracle Fusion Middleware Extensions for Applications libraries.
51.3.1.1 Initializing Sessions

Because it is not possible to authenticate users in the PLSQL layer, the API to initialize a session in PS/SQL is only expected to be called for testing. To use sessions, you must first configure your project to use application user sessions. For more information about configuring your project, see Configuring Your Project to Use Application User Sessions.

After you have configured your project to use application user sessions, you should be able to access sessions automatically if you are running a Java EE page.

For J2SE programs, such as JUnit tests, you must call an explicit API to initialize your session. As the following example shows, for JUnit tests in particular, this is most likely your setUp() or setUpBeforeClass() method along with a terminateSession call in the corresponding tearDown() or tearDownAfterClass() method.

Caution:

Remember, every call to initializeSession should have a corresponding terminateSession invoked after the code completes to prevent memory leaks.

Example 51-2 Initializing Your Session

```java
@BeforeClass
public static void setUpBeforeClass() throws exception {
    // Create a session for the OPERATIONS user
    List<String> roleGuids = new ArrayList<String>(1);
    List<String> roleNames = new ArrayList<String>(1);
    roleGuids.add("1807EDD02DBB11DDBFDC91643D402C34");
    roleNames.add("operationsRole");
    ApplSession session = ApplSessionUtil.initializeSession("43B84790D5F011DCAF4F5FFD8462C8E7", "OPERATIONS", roleGuids, roleNames, null);
}

@AfterClass
public static void tearDownAfterClass() throws exception {
    // note that if a connection to the 'initializeSession' call had been passed
    // in, it would would have to be freed here. Since null is passed in, the
    // connection that was obtained in that call will be freed automatically.
    ApplSessionUtil.terminateSession();
}
```

51.3.1.2 Getting Context Attributes

Accessing a context attribute is simple. First, make sure your project is configured to use application user sessions and then import the ApplSession and ApplSessionUtil classes. As the following example shows, after you complete
those tasks you can access the session and its properties using the static APIs that are provided.

Using the example, guid1 and guid2 should both return the same value. The ApplSessionUtil API is a convenience method that essentially calls the same code as the first two lines. One difference is that ApplSessionUtil.getUserGuid() raises an exception if the session is not available. This is true for all the ApplSessionUtil get methods, except for getSession(), which just returns null if there is no session.

All of the centrally maintained attributes listed above have corresponding get APIs available. The following example shows a mechanism for getting generic attributes. The following example shows the API you use to fetch attributes from a particular namespace.

**Example 51-3   Accessing the Session**

```java
ApplSession session = ApplSessionUtil.getSession();
String guid1 = session.getUserGuid();

String guid2 = ApplSessionUtil.getUserGuid();
```

**Example 51-4   Getting Generic Attributes**

```java
String attr1 = ApplSessionUtil.getSessionAttribute("ATTRIBUTE1");
```

**Example 51-5   Fetching Attributes From A Particular Namespace**

```java
String attr1 = ApplSessionUtil.getNamespaceAttribute("MY$NAMESPACE", "ATTRIBUTE1");
```

### 51.3.1.3 Setting Context Attributes

In addition to providing getters for all of the context attributes listed above, there are corresponding set APIs directly available in the ApplSession class. Attributes like the user name or the language are set automatically on the context at creation time, but the set APIs can also be called if an attribute must be changed in the middle of the request.

The following example sets the PRODUCT_FAMILY attribute to FND and also sets a generic attribute called ATTRIBUTE1 to VALUE1 on both the session and a private namespace using the Java APIs.

**Caution:**

Sets of ApplSession attributes get cached in the middle tier, and only written to the database when the session is detached or the ApplSession.synchronize() method is explicitly called. If the set operation takes place from within a request, synchronization will happen automatically. However, if you are running standalone java or need the attributes to get written to the database immediately, you should add a call to session.synchronize().

**Example 51-6   Setting Context Attributes**

```java
ApplSession session = ApplSessionUtil.getSession();
if (session != null)
{
    session.setProductFamily("FND");
    session.setSessionAttribute("ATTRIBUTE1", "VALUE1");
```
session.setNamespaceAttribute("MY$NAMESPACE", "ATTRIBUTE1", "VALUE1");
}

### 51.3.1.4 Accessing the Connection

The applications context does not hold onto connections, instead it obtains and releases them as needed. As the following example shows, if your application explicitly obtains a connection via the `ApplSession.getConnection()` API, you will need to add a `finally` block that releases that connection. It is recommended that you call the newly provided `ApplSession.releaseConnection(Connection conn)` API as it takes care of clearing out session-specific PL/SQL state in the connection before closing it.

**Example 51-7  Accessing the Connection of the Current ApplSession**

```java
Connection conn = null;
ApplSession session = ApplSessionUtil.getSession();
if (session != null)
{
  try
  {
    conn = session.getConnection();
    ...
  }
  finally
  {
    if (conn != null)
    {
      session.releaseConnection(conn);
    }
  }
}
```

### 51.3.1.5 Access Session Context Using the Java API

To access the context in your Java code, just call any of the static methods in the `ApplSessionUtil` class. If you are running from an environment where application user sessions are enabled, there should not be anything else you must do aside from importing the `ApplSessionUtil` class.

**Tip:**

If you are running without application user sessions enabled, an exception will be thrown when any of the above calls are made except the `getSession()` API. This API returns a `null` if sessions are not enabled.

The following is a more complex example of how you might use this:

You have a view object (`TestVO`) where you want to always display the current user name as one of the fields.

**To always display the current user name as one of the fields:**

1. Add a non-column based `UserName` attribute to the `TestVO` object.
2. Generate the View Row Class for the view object.
3. Look for the definition of `getUserName()`. As shown in the following example, change it to return the value of the call to `ApplSessionUtil.getUserName()` in the `TestVORowImpl.java` that gets autogenerated.
Whenever the TestVO view object is displayed, by default it will include the current user name field.

**Example 51-8 Changing the getUserName() Value**

```java
public String getUserName()
{
    return ApplSessionUtil.getUserName();
}
```

**Example 51-9 TestVO Example**

```java
public String getSysAdminInfo()
{
    String productName = ApplSessionUtil.getProduct();
    if ("FND".equals(productName))
    {
        return (String) getAttributeInternal(SYSADMININFO);
    }
    else
    {
        return null;
    }
}
```

The following example uses the SysadminInfo field that was added to the TestVO view object to display a value when running the FND product.

### 51.3.2 How to Access Sessions Using PL/SQL APIs

The applications context can also be accessed through APIs that are provided in the FND_GLOBAL package. As in Java, functions exist to get and to set each of the core attributes listed in Accessing Properties of the Applications Context, assuming you have initialized the connection to use sessions properly. For detailed information about the FND_GLOBAL package, see the javadoc.

#### 51.3.2.1 Initializing Sessions

The FND_GLOBAL.INITIALIZE_SESSION takes in the user GUID, the user name, and two lists of roles. The first represents the list of role GUIDs, and the second represents the list of corresponding role names. As the following example shows, the lists must be of the same length.

**Example 51-10 Initializing Sessions**

```sql
DECLARE
    l_roleguids FND_TABLE_OF_VARCHAR2_4000 := FND_TABLE_OF_VARCHAR2_4000();
    l ролenames FND_TABLE_OF_VARCHAR2_4000 := FND_TABLE_OF_VARCHAR2_4000();
BEGIN
    l_roleguids.extend(1);
    l ролenames.extend(1);
    l_roleguids(1) := '1807EDD02DBB11DDBFDC91643D402C34';
    l ролenames(1) := 'operationsRole';
    fnd_global.initialize_session('43B84790D5F011DCAF4F5FFD8462C8E7','OPERATIONS',
    l_roleguids,  l ролenames);
    <your code here>
    fnd_global.terminate_session;
END;
```
51.3.2.2 Getting Context Attributes
As an example, you can retrieve the current user by calling FND_GLOBAL.USER_NAME, and you can get a generic attribute by calling FND_GLOBAL.GET_SESSION_ATTRIBUTE.

51.3.2.3 Setting Context Attributes
As an example, you can set the language by calling FND_GLOBAL.SET_LANGUAGE, and you can set a generic attribute by calling FND_GLOBAL.SET_SESSION_ATTRIBUTE.
This chapter describes how to enforce security to authorize access to securable application artifacts created using Oracle Application Development Framework (Oracle ADF) in Oracle Fusion applications.

This chapter contains the following sections:

- Introduction to Function Security
- Function Security Implementation Process Overview
- Adding Function Security to the Application
- Learning More About Function Security in Oracle Fusion Applications

### 52.1 Introduction to Function Security

An important principle of Oracle Fusion function security ensures that end users do not have unintended access to web pages and application resources in an application that is created using Oracle Application Development Framework (Oracle ADF).

To enable access to application resources, you can use JDeveloper to create security policies to specify “who can perform what operations on what specific application artifacts.”

To create the security policy, you must consider the additional duties the end users of the application will perform and then grant the desired roles specific rights to:

- Access application resources, such as a custom ADF task flow that supports the duty
- Initiate only those operations on the data required by the duty

---

**Note:**

Securing the data of the application requires creating data security policies. For details about creating data security policies, see Implementing Oracle Fusion Data Security.

Function security controls access to securable ADF artifacts including ADF task flows and top-level web pages backed by ADF page definition files. Function security may also control the visibility of UI components within the web pages. Users who do not have the required privilege cannot view the task flow or view buttons that initiate create, update, and delete operations. For example, in a sales organization, duties such as Manage_Accounts and Manage_Invoices exist for roles, such as Sales_Manager or Sales_Associate. A function security policy might give end users who belong to the Sales_Manager role the ability to view and edit customer
invoices. Whereas, end users who do not belong to the Sales_Manager role, may enter the task flow to view customer invoices but not edit them.

52.1.1 Function Security Development Environment

Before you can implement function security for custom application resources, a security administrator must export the function security definitions from an LDAP-based policy store (typically from a staging environment) into a file-based policy store that you can work with in JDeveloper. The exported file will contain the function security artifacts that will enable you to run your application and access the resources that may otherwise have been secured by predefined function security policies.

The file you receive is formatted as XML and named jazn-data.xml. The XML definitions of the exported file comprise two major sections: an identity store to define valid end users of the application and a policy store to define the security policies that are specific to the application. Initially, only the policy store will be populated with security artifacts from the LDAP stores. The exported jazn-data.xml file will not contain the end user identities of the enterprise, thus the identity store section will initially appear empty.

Note:

As an Oracle Fusion security guideline, do not modify the predefined function security definitions contained in the jazn-data.xml file. Predefined security definitions include the security definitions of the Oracle security reference implementation and must not be modified. The security reference implementation provides role-based access control in Oracle Fusion Applications, and is composed of predefined security policies that protect functions, data, and segregation of duties. You should always add custom application roles to grant access rights. For details about restrictions when working with the file-based policy store, see What You May Need to Know About Actions That Developers Must Not Perform.

When you define security policies for custom Oracle ADF application artifacts, you should only use Oracle JDeveloper tools to work on the exported file-based policy store, and you should not edit the security definitions directly. JDeveloper supports iterative development of security so you can easily create, test, and edit security policies that you define.

After you customize security, you use JDeveloper to add end user identities to the identity store of the exported file before running and testing the application in JDeveloper's Integrated WebLogic Server. You provision a few test end user identities by defining user groups and then assign those groups to application roles to simulate how the actual end users of the enterprise will access the secured application artifacts. When you deploy the application in your development environment, JDeveloper migrates the identity store you created to the embedded LDAP of Integrated WebLogic Server. The application policy store is migrated to a system-jazn-data.xml file that aggregates the security policies definitions of all applications in your workspace.

After testing in JDeveloper, you must consult with the security administrator to merge the LDAP-based application policy store in the staging environment with the security policies that you added to the exported jazn-data.xml file. The staging environment is an LDAP-based Oracle WebLogic Server configured to use Oracle Internet Directory (OID) for the enterprise's application policy store and identity store.
(note that the stores of the staging server are LDAP-based and not file-based). Initially, the staging environment allows further testing using that server’s identity store before deploying to the production environment. Thus, end user identities created in JDeveloper are not migrated to standalone Oracle WebLogic Server and are used only in Integrated WebLogic Server to test the extended application.

52.1.2 Function Security Implementation Scenarios

As an Oracle Fusion security guideline, when you secure the functions of your application, you should not modify the predefined security definitions specified by the Oracle Fusion security reference implementation. When you modify the file-based policy store, always create new security definitions.

---

**Note:**

The term *protected* in this chapter refers to the default Oracle Fusion application condition that denies end users access to database resources and application artifacts. In contrast, the term *secured* refers to resources that have been made accessible to end users through security policies created for this purpose. Therefore, a security policy specifically *enables access* to the resource based on the access rights it confers to the end user.

---

The following table summarizes the function security scenarios that Oracle Fusion security supports. The "Application Developer Tasks" column of the table provides a brief description of the security artifacts involved in each scenario.

**Table 52-1  Oracle Fusion Function Security Use Cases**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Control whether the end user associated with a particular role may access a new task flow and view all the web pages of the flow.</td>
<td>Create a new entitlement grant. The new task flow will be inaccessible by default (also called <em>protected</em>) and will require a new function security policy to grant end users access. Because the end user duty being secured is rarely addressed by grants to a single resource, the Oracle Fusion security best practice is to create entitlement grants. Entitlement grants provide the means to aggregate multiple securable resources into a named security group so that privileges for the entire group can be granted to application roles through a single statement.</td>
<td>Enable ADF Security on the user interface project to protect all task flows (and the web pages they contain). Then, in the file-based policy store, create a resource definition for the task flow and assign the definition as a member of an entitlement (defined in the policy store as a <em>permission set</em>) that you name. Then, create the security policy by granting the entitlement to a custom application role that you either created or consulted with a security administrator to create for you. For more information, see How to Create Entitlement Grants for Custom Application Roles.</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Control whether the end user associated with a particular role may access a new top-level web page. In Oracle Fusion applications, a top-level web page is one that is not contained by a task flow.</td>
<td>Create a new entitlement grant. The new top-level web page will be inaccessible by default (also called protected) and will require a new function security policy to grant end users access. The ability to secure individual web pages in Oracle Fusion applications is reserved for top-level web pages backed by an ADF page definition file only.</td>
<td>Enable ADF Security on the user interface project to protect all top-level web pages backed by ADF page definition files. Then, in the file-based policy store, create a resource definition for the web page and assign the definition as a member of an entitlement (defined in the policy store as a permission set) that you name. Then, create the security policy by granting the entitlement to a custom application role that you either created or consulted with a security administrator to create for you. For more information, see How to Create Entitlement Grants for Custom Application Roles.</td>
</tr>
<tr>
<td>Control whether a new task flow or a new top-level web page is publicly accessible.</td>
<td>Create a new resource grant. The new ADF artifact will be inaccessible by default (also called protected) and will require a new function security policy to grant end users access. Because the publicly accessible artifact is a single resource, the Oracle Fusion security best practice is to create resource grants (rather than entitlement grants) for publicly accessible artifacts.</td>
<td>Enable ADF Security on the user interface project to protect all ADF-backed application artifacts. Then, in the file-based policy store, grant an action (defined in the policy store as a permission) directly to the artifact. Then, create the security policy by granting the permission to a built-in OPSS application role. For more information, see How to Define Resource Grants for OPSS Built-In Roles.</td>
</tr>
<tr>
<td>Determine whether the end user associated with a particular role has the right to view UI components, such as buttons that initiate create, update, or delete operations in the displayed web page.</td>
<td>Create a grant. Access to user interface components, such as buttons can be controlled by rendering the button in the user interface based on a permission check.</td>
<td>Conditionally render the component by entering ADF Security Expression Language (EL) utility methods on the rendered attribute of the button to test whether the end user has the necessary permission grant. For more information about rendering components using EL utility methods, see the &quot;Enabling ADF Security in a Fusion Web Application&quot; chapter of the Developing Fusion Web Applications with Oracle Application Development Framework.</td>
</tr>
</tbody>
</table>

For more information about how data security and function security work together to control access to the data and functions of the application, see the "Customizing Security for Oracle ADF Application Artifacts" chapter in the Extensibility Guide for Developers.

### 52.1.3 Function Security-Related Application Files

When you create a Fusion web application, JDeveloper creates specific files that are needed to secure the application. Additionally, when you run the Configure ADF
Security wizard, JDeveloper updates these files to reflect the selections you make in the wizard.

For details about the created files, see the "What Happens When You Enable ADF Security" section in the Developing Fusion Web Applications with Oracle Application Development Framework.

52.2 Function Security Implementation Process Overview

An ADF bounded task flow that you add to your application is one of the main ADF artifacts that you can secure. You can also directly secure top-level web pages that are backed by ADF page definitions to specify data bindings. Although you can secure a variety of application resources, implementing function security follows a similar pattern.

Before You Begin

You will need to complete these tasks:

1. Decide the names of custom application roles that your application will specify as the grantee of security privileges.

2. Optionally, ask the security administrator to create custom application roles with the names you supply.

   If a security administrator creates the application roles you identify, then those custom application roles will already appear in the policy store section of the exported jazn-data.xml file.

   If you do not ask the security administrator to create custom application roles, then you must create them in JDeveloper before you define security policies.

3. Consult a security administrator to export all predefined function security policies of the application that you are customizing into a jazn-data.xml file.

   For details about extracting data from an OID store to a file, see the "Security Store Management" section of the Administrator’s Guide.

To implement function security:

1. Copy the exported jazn-data.xml file into your application workspace.

   This is the file that JDeveloper will update when you create function security policies. In order for JDeveloper to use the file, copy the file to your application workspace in the <JDevAppHome>/src/META-INF folder.

2. Determine which ADF artifacts should be secured and grant entitlement privileges to custom application roles to specify the access rights of end users.

   Although ADF Security permits you to define function security policies for ADF artifacts using only resource privilege grants, an Oracle Fusion security best practice is to define access policies using entitlement grants except for publicly accessible application artifacts.

   For details about securing application functions, see How to Create Entitlement Grants for Custom Application Roles.

3. Determine which ADF artifacts should be public and grant resource privileges to an appropriate OPSS built-in application role.

   For details about making application functions public, see How to Define Resource Grants for OPSS Built-In Roles.
4. Opt into the previously defined function security policies by running the Configure ADF Security wizard to enforce OPSS authorization checking.

   For details about enabling security on the user interface project, see How to Enforce Authorization for Securable ADF Artifacts.

5. Determine which user interface components you want to associate with user entitlements, and enter ADF Security-based EL utility methods on the component to make it logically consistent with its target.

   ADF does not enforce security on user interface components, such as buttons or links that navigate to securable artifacts (pages and task flows).

6. Define test user identities and run the application in JDeveloper to simulate the privileges of the enterprise users who will eventually interact with the application.

   For details about adding test user in JDeveloper, see How to Enable Authentication and Test the Application in JDeveloper.

To migrate security policies to the domain policy store:

1. After testing is complete, remove the test users from the jazn-data.xml file and provide the updated jazn-data.xml file to the security administrator to merge the file-based policy store with the application policy store in the staging environment.

   JDeveloper must not be used as an identity store provisioning tool, and you must be careful not to deploy the application with user identities that you create for testing purposes. Deploying user identities with the application introduces the risk that malicious users may gain unintended access.

2. The security administrator provisions enterprise users by mapping enterprise roles defined in the staging environment identity store to the custom application roles.

   For information about how the security administrator provisions enterprise users using Oracle Authorization Policy Manager, see the product-specific security guides.

3. Before running the application in the staging environment, the security administrator must reconcile the application roles GUIDs of any data security policies that were created based on new custom application roles.

   When the file-based policy store is merged, the GUIDs of application roles are not preserved.

4. Continue testing the application in the staging environment before deploying the application to production and merging the policies into the production environment application policy store.

   For details about how to modify the application to use the identity store and policy store of the staging environment, see What You May Need to Know About Testing.

52.3 Add Function Security to the Application

You implement function security by identifying the type of resource that corresponds to the ADF artifact whose function you intend to secure. You then select a resource instance of that type and select the action that corresponds to the artifact function you intend to grant to end users. Function security aggregates these resource / action pairs as an entitlement that serves as the grantable entity. Each entitlement can include as
many resource / action pairs as needed to describe a particular duty to be performed by the end user. To support this goal, entitlements can include resources of different types. To create a function security policy, you then grant the entitlement to a custom application role, also called a duty role in Oracle Fusion applications.

In cases where a resource should be publicly accessible, you will not need to aggregate multiple resources to define a particular duty. Instead, you can create a resource-based function security policy with a single resource /action pair defined as the grantable entity.

---

**Note:**

Although ADF Security permits you to define function security policies for ADF artifacts using only resource privilege grants, an Oracle Fusion security best practice is to define access policies using entitlement grants except for publicly accessible application artifacts.

---

To simplify the task of securing the functions of your application, ADF provides the ADF Security framework. ADF Security defines a containment hierarchy that lets you define a single security policy for the ADF bounded task flow and its contains web pages. In other words, the security policy defined at the level of the bounded task flow, secures the flow’s entry point and then all pages within that flow are secured by the same policy. For example, a series of web pages may guide new end users through a registration process and the bounded task flow controls page navigation for the process.

Specifically, ADF artifacts that are automatically protected by ADF Security and require function security policies to grant users access are:

- ADF bounded task flow protects the entry point to the task flow, which in turn controls the end user’s access to the pages contained by the flow

  The ADF unbounded task flow is not a securable application artifact and thus does not participate in OPSS authorization checks. When you need to secure the constituent pages of an unbounded task flow, you define policies for the page definition files associated with the pages instead.

- ADF page definition files associated with top-level web pages and regions

  For example, a page may display a summary of products with data coordinated by the ADF bindings of the page’s associated ADF page definition file.

---

**Best Practice:**

Do not create entitlement grants for the individual web pages of an ADF bounded task flow. When the end user accesses the bounded task flow, security for all pages will be managed by the entitlements you create for the task flow. This supports a well-defined security model for task flows that enforces a single entry point for all end users. For additional best practice information about ADF and function security, see What You May Need to Know About ADF Security Guidelines.

---

To add function security to the application:
1. Determine which ADF task flows and top-level web pages should be secured and grant entitlement privileges to custom application roles to define the duties of end users.

2. Determine which ADF task flows and top-level web pages should be public and grant resource privileges to an appropriate OPSS built-in application role.

3. Determine which user interface components should be secured and create a resource permission and grant the view action in the entitlement grant to a custom application role.

ADF does not automatically enforce security on user interface components, such as buttons or links that navigate to securable artifacts (pages and task flows). To secure these application artifacts, you must create a resource permission that the application can test by entering ADF Security-based EL utility methods on the component to make it logically consistent with its target.

4. Opt into the previously defined function security policies by running the Configure ADF Security wizard to enforce OPSS authorization checking.

5. Define test user identities and run the application in JDeveloper to simulate the privileges of the enterprise users who will eventually interact with the application.

### 52.3.1 How to Create Entitlement Grants for Custom Application Roles

Because the particular end user duty you want to secure is rarely addressed by grants to a single resource, you define the access policy for securable ADF artifacts by creating entitlement grants. Resource grants are only necessary to define publicly accessible application artifacts, as described in How to Define Resource Grants for OPSS Built-In Roles.

You create entitlement grants in the Entitlements Grants page of the jazn-data.xml file overview editor. The grants you create will appear as metadata in the policy store section of the jazn-data.xml file. This metadata defines an entitlement (identified in the XML definition as `<permission-set>`) comprised of resource instance / action pairs that you select. This entitlement is a grantable entity that you then grant to a custom application role.

**Note:**

Note that, in Oracle Fusion Applications, application roles are called duty roles. You must not modify predefined duty roles, and you must always add custom application roles (duty roles) to grant access permissions.

The list of resource types appears in the security policy overview editor. The resource type you select filters the resource instances defined within the projects of your application’s workspace. The resource type selection also determines the list of available actions displayed by the overview editor. For example, when you select the Task Flow resource type, the overview editor will display all of the task flows in the user interface projects that you select and also displays the view action that you can associate with the available ADF bounded task flow resources.

The following table lists the resource types displayed in JDeveloper and identifies the associated resource and actions.
Table 52-2  Resource Types of Securable ADF Artifacts

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Supports These Resources and Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADF Task Flow</td>
<td>Defines personalize, customize, grant, edit, and view actions on ADF bounded task flows in a Fusion Web application.</td>
</tr>
<tr>
<td>ADF Region</td>
<td>Defines personalize, customize, grant, edit, and view actions on regions and web pages backed by an ADF page definition file in a Fusion Web application.</td>
</tr>
<tr>
<td>ADF Entity</td>
<td>Not used by Oracle Fusion Applications. Data security is provided by Oracle Fusion Data Security, as described in Implementing Oracle Fusion Data Security.</td>
</tr>
<tr>
<td>ADF Method</td>
<td>Defines execute, invoke, and view actions on ADF methods in a Fusion web application.</td>
</tr>
<tr>
<td>Resource</td>
<td>Defines custom actions on any functional artifact in a Fusion web application. Resource permissions are useful to create grants for UI components, as described in How to Protect UI Components.</td>
</tr>
<tr>
<td>Webservice</td>
<td>Defines invoke actions on Fusion Web services. For more details about securing Web services, see Authorizing the Web Service with Entitlement Grants.</td>
</tr>
</tbody>
</table>

To define an entitlement grant for a securable ADF artifact, use the Entitlement Grants page of the overview editor for the jazn-data.xml file. This editor is also called the security policy overview editor.

Before you begin:

It may be helpful to understand the details of the ADF Security containment model. For more information, see What You May Need to Know About ADF Security Guidelines.

You will need to complete these tasks:

- Consult the security administrator for the enterprise to obtain the jazn-data.xml file that contains the predefined function security policies for your application. You must add the file to your application workspace, as explained in Function Security Implementation Process Overview.


To define an entitlement grant for an ADF artifact:

1. In the main menu, choose Application and then Secure > Entitlement Grants.

2. In the Entitlement Grants page of security policy overview editor, click the Add Entitlements icon in the Entitlements list.

The overview editor displays all the resources that your application defines.
3. In the Resources section, click the Add Member Resource icon to add a member resource to the entitlement.

4. In the Select Resources dialog, select the resource from the Resource Type dropdown and then select the desired project in the Source Projects list.

   The dialog displays all the projects in your application workspace.

5. In the Available Resources list, select the resource from and click the Add icon.

   The dialog displays all the resources defined by your selected project.

6. In the Actions lists, select the desired action for the selected resource.

   The following figure shows the overview editor with the View action selected for the task flow and added to MyEntitlement.

   Figure 52-1 Adding a Bounded Task Flow as a Resource in an Entitlement Grant

7. Add other desired resources to the list.

8. In the Grants section of the security policy overview editor, click the Add Role Grants icon to grant the entitlement to an application role.

9. In the Select Application Roles dialog, select one or more custom application roles.

   The dialog displays all the application roles from the jazn-data.xml file. You must not add a grant to a predefined application role (also called duty roles in the terminology of Oracle Fusion Applications). Only select custom application roles that either you created in JDeveloper or that were created by a security administrator for this purpose.

10. Click OK.

11. You can repeat these steps to add other resources and make grants on those resources to the same entitlement for the same custom application role.
52.3.2 What Happens After You Create an Entitlement Grant

When you use the security policy editor in JDeveloper to create an entitlement grant, JDeveloper modifies the source for the application policy store in the jazn-data.xml file. The policy store section of the file contains a <resource-type> definition (that identifies the actions supported for resources of the selected type), a <resource> definition (to identify the resource instance that you selected from your application and mapped to a resource type), a <permission-set> definition (to define the resources and actions to be granted as an entitlement), and a <grant> definition with one or more entitlements (defined in the XML as a permission set) granted to the desired application roles (the grantee).

As the following example shows, entitlement-based security policies in the Oracle Fusion application are defined in the <jazn-policies> element and consist of one or more entitlements granted to a single application role.

Example 52-1 Entitlement-Based Policy Definition in the jazn-data.xml File

```xml
<?xml version="1.0" ?>
<jazn-data>
  <policy-store>
    <applications>
      <application>
        <name>MyApp</name>
        <app-roles>
          <app-role>
            <name>AppRole</name>
            <display-name>AppRole display name</display-name>
            <description>AppRole description</description>
            <guid>F5494E409CFB11DEBFEBC11296284F58</guid>
            <class>oracle.security.jps.service.policystore.ApplicationRole</class>
          </app-role>
        </app-roles>
        <role-categories>
          <role-category>
            <name>MyAppRoleCategory</name>
            <display-name>MyAppRoleCategory display name</display-name>
            <description>MyAppRoleCategory description</description>
          </role-category>
        </role-categories>
        <!-- resource-specific OPSS permission class definition -->
        <resource-types>
          <resource-type>
            <name>APredefinedResourceType</name>
            <display-name>APredefinedResourceType display name</display-name>
            <description>APredefinedResourceType description</description>
            <provider-name>APredefinedResourceType provider</provider-name>
            <matcher-class>oracle.security.jps.ResourcePermission</matcher-class>
            <actions-delimiter>,</actions-delimiter>
            <actions>write,read</actions>
          </resource-type>
        </resource-types>
        <resources>
          <resource>
            <name>MyResource</name>
            <display-name>MyResource display name</display-name>
          </resource>
        </resources>
    </application>
  </applications>
</jazn-data>
```
52.3.3 How to Define Resource Grants for OPSS Built-In Roles

A common requirement of the application is that some web pages be available to all end users, regardless of their specific access privileges. For example, the home page should be seen by all visitors to the site, while a corporate site should be available only to those who have identified themselves through authentication.

In both cases, the page may be considered public, because the ability to view the page is not defined by the end users’ specific privileges. Rather, the difference is whether the end user is anonymous or a known identity.
In the OPSS security model, you differentiate between the absence of security and public access to content by granting access privileges to the anonymous-role principal. The anonymous-role allows access to a resource by unauthenticated users, for example, guest users. To provide access to authenticated users only, the policy must be defined for the authenticated-role principal.

The built-in OPSS role authenticated-role stands for any authenticated user and is useful to implement authorization checks for end users who do not need to be explicitly assigned to specific custom application roles to get access to a resource. The authenticated-role can be directly granted any resource grants.

Before you begin:

It may be helpful to have an understanding of OPSS support for public, unprotected resources. For more information, see the “Understanding Users and Roles” chapter of the Oracle Fusion Middleware Applications Security Guide, in the Oracle Fusion Middleware Online Documentation Library.

It may also be helpful to understand the details of the ADF Security containment model. For more information, see What You May Need to Know About ADF Security Guidelines.

You will need to complete this task:

- Consult the security administrator for the enterprise to obtain the jazn-data.xml file that contains the predefined function security policies for your application. You must add the file to your application workspace, as explained in Function Security Implementation Process Overview.

To grant public access to securable ADF artifact:

1. In the main menu, choose Application and then Secure > Resource Grants.

2. In the Resource Grants page of security policy overview editor, select the resource from the Resource Type dropdown and then select the desired project in the Source Projects list.

   The overview editor displays all the projects in your application workspace.

3. In the Resources column, select the ADF artifact for which you want to grant access rights.

   Tip: Click the lock icon to show only those resources that do not yet have grants.

4. In the Granted to column, click the Add Grantee icon and choose Add Application Role.

5. In the Select Application Roles dialog, select one of these built-in application roles:

   - **anonymous-role** means the resource will be accessible to anyone who visits the site. A grant to this role is necessary if you want to make web pages backed by securable ADF artifacts accessible before an end user logs in. For example, you would grant to anonymous-role for an ADF bounded task flow that manages customer registration.

   - **authenticated-role** means the resource will be accessible only to authenticated users (ones who visit the site and log in). For example, you would grant to authenticated-role for an ADF bounded task flow that manages employee registration.
6. In the Select Application Roles dialog, click **OK**.

7. In the Resource Grants page of the overview editor, in the **Actions** column, select the desired action.

The following figure shows the overview editor with the View action selected for the task flow and granted to **authenticated-role**.

**Figure 52-2  Granting to authenticated-role in the Overview Editor**

---

52.3.4 What Happens When You Make an ADF Resource Public

When you use the security policy editor in JDeveloper to create a resource grant, JDeveloper modifies the source for the application policy store in the **jazn-data.xml** file.

The following example shows a resource-based security policy in the **jazn-data.xml** file that makes a customer registration task flow public to all authenticated users. The grant to the OPSS built-in role **authenticated-role** contains a single view permission for a bounded task flow, **customer-registration-task-flow**. With this grant, any authenticated user will be able to enter the employee registration task flow.

**Example 52-2  Resource-Based Policy Definition in the jazn-data.xml File**

```xml
<policy-store>
  ...
  <jazn-policy>
    <grant>
      <grantee>
        <principals>
          <principal>
            <class>oracle.security.jps.internal.core.principals.JpsAuthenticatedRoleImpl</class>
            <name>authenticated-role</name>
          </principal>
          ...
        </principals>
      </grantee>
      <permissions>
        <permission>
          <name>view</name>
          <resource>customer-registration-task-flow</resource>
        </permission>
      </permissions>
    </grant>
  ...
</jazn-policy>
</policy-store>
```
52.3.5 How to Protect UI Components

Security expressions in the ADF Security EL namespace let you check the user permission, authentication and role membership status before displaying UI components in your page. When you need fine-grained protection over UI components to secure functional parts of the application that are not part of task flow security or web page security, you use the EL value userGrantedResource and a resource grant that you define for a custom resource type. For instance, through a resource grant that you create for a menu item, the application may either enable or disable a Cancel Shipment menu item to limit who may perform customer order updates.

Resource grants for custom resources types rely on the ResourcePermission class, which is provided by Oracle Platform Security Services (OPSS) and packaged with the application.

You use the overview editor for security policies to create a security policy for the custom resource type. And, in your web page, you use the security expression to evaluate the user’s access rights to the UI component projected by the resource grant.

To grant resource policies for a custom resource type:

1. Create the custom resource type for the UI component you want to protect.
2. Create the resource grant for the custom resource.
3. Associate the rendering of a UI component with a role’s granted resource permission.

For complete details about protecting UI components, see the "How to Protect UI Components Using OPSS Resource Permissions and EL" section in the Developing Fusion Web Applications with Oracle Application Development Framework.

52.3.6 How to Enforce Authorization for Securable ADF Artifacts

You run the Configure ADF Security wizard to enable authorization and make the function security policies you define effective. When you run the Configure ADF Security wizard, it has the following affect:

- It configures your application to enable OPSS security when running in the JDeveloper test environment.
- It allows the Integrated WebLogic Server to use the file-based security policies to authorize access to application resources by the end user (where OPSS determines at runtime whether the end user (represented by the JAAS subject) has the privileges necessary to access the resources they intend.
When you run the wizard, you are effectively enforcing authorization checking for all securable ADF artifacts. The wizard also enables the ADF authentication servlet to require the end user to log in the first time a page in the application is accessed.

This configuration requires a valid user to access the pages of your application. This assumes that you will define custom application roles and assign explicit grants to those roles to manage access to securable ADF artifacts, as described in How to Create Entitlement Grants for Custom Application Roles. Alternatively, when you want to make a page public and accessible by unauthenticated user, you must explicitly grant to a built-in OPSS role, as described in How to Define Resource Grants for OPSS Built-In Roles.

Before you begin:

It may be helpful to have an understanding of the configuration changes made by the Configure ADF Security wizard. For more information, see the "Enabling ADF Security in a Fusion Web Application' chapter of the Developing Fusion Web Applications with Oracle Application Development Framework, in the Oracle Fusion Middleware Online Documentation Library.

It may also be helpful to have an understanding of the test-all role when you want to test the application without restricted access. For more information, see the "Enabling ADF Security in a Fusion Web Application’ chapter of the Developing Fusion Web Applications with Oracle Application Development Framework, in the Oracle Fusion Middleware Online Documentation Library.

To enforce authorization:

1. In the main menu, choose Application and then Secure > Configure ADF Security.
2. Complete the Configure ADF Security wizard pages.

For complete details about running the wizard, see the "Enabling ADF Security' section in the Developing Fusion Web Applications with Oracle Application Development Framework.

52.3.7 How to Enable Authentication and Test the Application in JDeveloper

Authentication is enabled when you run the Configure ADF Security wizard, as described in How to Enforce Authorization for Securable ADF Artifacts. This means you when you run your application, you will be prompted to log in upon accessing any page backed by securable ADF artifacts.

To test your application, you will need to create test user identities and provision them with the custom application roles that you defined. The end user’s membership in an application role defines their access privileges to the resources. If you prefer to be consistent with the Oracle Fusion standard for provisioning users and simulate how the actual end users of the enterprise access resources, you can optionally provision test users by defining enterprise roles consisting of groups of users (called job roles in Oracle Fusion applications) and then assign those groups to application roles (called duty roles in Oracle Fusion applications).

Note that if you enabled the option to grant to a test role when you run the Configure ADF Security wizard, your grants may be of the following two types:

- For the task flows that you create before running the Configure ADF Security wizard, the wizard automatically assigns them to the test-all role. You will need to remove these grants and create valid grants to your custom application
roles. For each entitlement granted to a specific role, the equivalent grant to the test-all role must be removed.

- The new task flows you create after running the Configure ADF Security wizard are not granted to the test-all role automatically. You need to manually grant the entitlements for the new task flows to your custom application roles.

Related Links
The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For details about creating test users, see the "Enabling ADF Security in a Fusion Web Application" chapter of the Developing Fusion Web Applications with Oracle Application Development Framework, in the Oracle Fusion Middleware Online Documentation Library.

52.3.8 What You May Need to Know About Actions That Developers Must Not Perform

Security definitions that are predefined in the Oracle Fusion security reference implementation must not be changed by developers. When modifying the file-based policy store, always create custom application roles and define new entitlement grants.

Specifically, developers must not make the following changes to the predefined security definitions of the Oracle Fusion security reference implementation.

- Add or remove entitlement grants on predefined application roles (those supplied by Oracle).
- Add or remove resources on predefined entitlements (those supplied by Oracle).
- Add or remove actions on resources on predefined entitlements (those supplied by Oracle).

For more information about the Oracle Fusion security reference implementation, see the product-specific security guides.

52.3.9 What You May Need to Know About Testing

When the application needs to undergo testing either in your local environment or on a staging server, the following changes will ensure that the application uses the LDAP-based identity and policy stores configured on the staging server.

To configure the deployed application to use the security repositories on the target server:

1. Set the application name in the web.xml file to point to the application stripe on the target server.

   The domain policy store is logically partitioned in stripes, one for each application name specified in the file system-jazn-data.xml under the element <applications>. Each stripe identifies the subset of domain policies pertaining to a particular application. For more details about application stripes, see the "Managing the Policy Store" chapter in the Oracle Fusion Middleware Applications Security Guide.

2. Set the jps.policystore.applicationid in the weblogic-application.xml file to point to the application stripe on the target server.
3. If you want the policies to persist on the staging server after the application is undeployed, then set the `jps.policystore.removal` flag in the `weblogic-application.xml` file.

4. Note that the `jps-config.xml` file in the application does not need to be modified. When the application is deployed, the staging server will have its own instance of the `jps-config.xml` file which is configured through a WLST command (the `reassociateSecurityStore` command). Therefore, the application `jps-config.xml` file can remain unchanged.

If you use JDeveloper to deploy to the application to standalone Oracle WebLogic Server, then you must ensure the Users and Groups checkbox is not selected in the application deployment properties (Menu Application > Secure > Configure Security Deployment...), as shown in the following figure.

**Figure 52-3  Application Deployment Properties Related to Security**

![Application Deployment Properties](image)

As a security best practice, you should not migrate users and groups that you create in JDeveloper. If the Users and Groups checkbox is selected, test users and groups in `jazn-data.xml` will be merged into the target server with different GUIDs than those used to grant data security privileges.

**Note:**

JDeveloper must not be used as an identity store provisioning tool, and you must be careful not to deploy the application with user identities that you create for testing purposes. Deploying user identities with the application introduces the risk that malicious users may gain unintended access. Instead, rely on the system administrator to configure user identities through the tools provided by the domain-level identity management system.

Before testing the application in the staging environment, any custom application roles that you created will need to be created in the LDAP application policy store. These
new application roles will receive new GUIDs and any data security policies defined for application roles of the same name must have their GUIDs reconciled.

Related Links
The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For details about the LDAP identity and policy store configuration, see the "OPSS Services" part of the Oracle Fusion Middleware Applications Security Guide, in the Oracle Fusion Middleware Online Documentation Library.
- For details about how security administrators reconcile GUIDs in the data security repository, see the "Reconciling GUIDs" section in the Administrator's Guide.

52.3.10 What You May Need to Know About ADF Security Guidelines

ADF implements a particular security model. Follow these rules to address problems you encounter when adding security to the application:

- Bounded task flows are secured by default. Secured by default means OPSS will check authorization on all bounded task flows when ADF Security has been enabled.
- Pages and page fragments backed by an ADF page definition file are also secured by default when not embedded in a bounded task flow. Keep in mind that OPSS will not check authorization for pages or page fragments that do not have a corresponding ADF page definition file.
- Pages and page fragments embedded in bounded task flows will not be checked by OPSS for authorization. Instead, they are granted or denied as a unit depending on the entitlement granted to the bounded task flow. That means those pages and page fragments do not need to be explicitly granted in the policy store.
- Bounded task flows embedded in bounded task flows will be checked by OPSS for authorization.
- ADF does not enforce security on user interface components, such as buttons or links) that navigate to securable artifacts (pages and task flows). An explicit EL expression must be attached to the component to make it logically consistent with its target.
- The test-all role is just a means of not breaking the application when ADF Security is enabled. Therefore, it should never be deployed, since it grants access to the application for non-authenticated users.

Related Links
The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For information about ADF application artifacts, see the Oracle Fusion Middleware Developing Fusion Web Applications with Oracle Application Development Framework guide, in the Oracle Fusion Middleware Online Documentation Library.
52.4 Learning More About Function Security in Oracle Fusion Applications

The following documents within the Oracle Fusion Applications Online Documentation Library provide details about the supporting tasks of the Oracle Fusion Applications security administrator.

- For information about how the security administrator creates application roles using Oracle Authorization Policy Manager, see the product-specific security guides.

- For information about how the security administrator exports the application policy store, see the "Extracting Data from an OID Store to a File" section in the Administrator’s Guide.

- For information about how the security administrator merges the policies using Oracle Authorization Policy Manager, see the "Upgrading Fusion Application Policies" section in the Administrator’s Guide.

- For information about how the security administrator reconciles GUIDs using a WLST command, see the "Reconciling GUIDs" section in the Administrator’s Guide.

Related Links

The following documents provide additional information about Oracle Fusion Middleware topics discussed in this chapter:

- For information about OPSS (Oracle Platform Security Services) and application roles, see the "Understanding Security Concepts" part in the Oracle Fusion Middleware Applications Security Guide, in the Oracle Fusion Middleware Online Documentation Library.

- For information about ADF application artifacts, see the Oracle Fusion Middleware Developing Fusion Web Applications with Oracle Application Development Framework guide, in the Oracle Fusion Middleware Online Documentation Library.

- For details about creating test users, see the Enabling ADF Security in a Fusion Web Application section of the Oracle Fusion Middleware Developing Fusion Web Applications with Oracle Application Development Framework guide, in the Oracle Fusion Middleware Online Documentation Library.

- For details about using EL utility methods to render UI components based on user roles, see the "Enabling ADF Security in a Fusion Web Application" chapter of the Developing Fusion Web Applications with Oracle Application Development Framework, in the Oracle Fusion Middleware Online Documentation Library.
This chapter describes the best practices for securing Web services in an Oracle Fusion application using an Oracle Web Services Manager (Oracle WSM) feature called global policy attachments (GPA).

This chapter contains the following sections:

- Introduction to Securing Web Services Use Cases
- Understanding Best Practices
- Attaching Policies Globally
- Attaching Policies Locally
- Authorizing the Web Service with Entitlement Grants
- What Happens At Runtime: How Policies Are Enforced
- Maintaining Application Session Context Across Web Service Requests

### 53.1 Introduction to Securing Web Services Use Cases

The service use case patterns described in Common Service Use Cases and Design Patterns must be secured. Security requirements vary depending on service and client implementations.

The following Web service integration scenario represents a typical integration pattern. The scenario highlights the security implementation requirements of each service component, as illustrated in the accompanying figure below.

**An example Web service integration scenario that relies on Oracle Mediator:**

1. A user logs into an ADF web application.
2. The application raises a business event.
3. The business event triggers an SOA composite through an Oracle Mediator service component.
4. The Mediator component invokes a Business Process Execution Language (BPEL) process.
   - The Mediator can invoke various BPEL processes based on the incoming event. It can also transform the event to the payload that BPEL process takes.
5. The BPEL process interacts with ADF Business Components Web services so as to execute the business logic of the use case.

While event generation typically begins with ADF Business Components, as an alternative, you can also trigger or call a SOA composite and either directly or
indirectly invoke the ADF Business Components Web service by any one of the following:

- Another SOA composite through a business event.
- Java or PL/SQL code through a business event.
- Synchronously calling the SOA composite through a JAX-WS proxy.
- An ADF Web service data control.

These alternative flow patterns require different security implementations. For example, both JAX-WS proxy and ADF Web service data control can access the client security policy references in the `connections.xml` file of the application to propagate the current user. However, when the event is triggered from the user interface and ADF Business Components, the ADF Business Components Web service can be invoked synchronously using the `ServiceFactory` interface (using either RMI or SOAP).

The following figure illustrates the possible event generation scenarios for Oracle Fusion applications. The main scenario flow—ADF Business Components-generated events—is illustrated in the center, along with numbers (enclosed in circles) illustrating the corresponding steps of the above integration scenario. Possible alternative flows are represented by dashed lines and numbers.

**Figure 53-1  Sample Web Service Integration Scenario**

Oracle Fusion applications typically use SOAP services. Use Oracle Web Services Manager (Oracle WSM) to secure these services. Following are the main recommendations when using Oracle WSM with Oracle Fusion Applications.
• Attach Oracle WSM authentication service policies to Web services and BPEL process Web service bindings only when it is not possible to globally attach policies at the domain level.

• Attach Oracle WSM client policies to Web services references in BPEL Partner links, proxies and ADF Web services data controls.

• It is a requirement to enforce authentication on all ADF Business Components Web services and exposed BPEL processes. Global policy attachment automatically satisfied this requirement.

• It is a requirement to enforce authorization on all ADF Business Components Web services.

• No authorization is required for SOA components, although it is possible to implement authorization checks for users and enterprise roles. You can enable authorization checks in ADF Business Components Web services.

• All request and response messages must be protected for integrity and confidentiality using either message protection security policies or message protection and authentication with SSL security policies. Oracle WSM transparently handles this.

• Oracle WSM policies do not apply to events.

53.2 Understanding Oracle Web Services Manager Best Practices

You can secure Web services and clients used in your Oracle Fusion application with Oracle Web Services Manager (Oracle WSM). A component of Oracle SOA Suite, Oracle Web Services Manager provides security policies that you can declaratively attach to SOAP services and clients.

Oracle Fusion Applications make use of an Oracle WSM feature called global policy attachments (GPA). Using GPA, policies are not attached locally, but are specified at a global level. At runtime, components simply inherit the global policy and Oracle WSM enforces it.

Unlike local policy attachments (LPA), which need to be added at every Web service client and server, global policy attachment (GPA) can be attached at a domain level. This makes it easy for the system administrator to have a uniform policy for all Web services across the domain.

Note:

All Oracle Fusion application Web services should use global policy attachment wherever possible.

Certain scenarios exist in which GPA cannot be used:

• Public Web services (those that do not need any user authentication) should use LPA to locally attach a "no authentication" policy on both the client side and the service side.

• When a Web service client needs to connect to a service using a particular user name and password, you need to specify the user name and password using a configuration override.
But GPA policies do not allow configuration overrides, which means you must use LPA to attach a username password policy on the client side. Note that even though configuration overrides require that you implement LPA on the client side, you still can allow the system administrator to define GPA on the server side for username password policies. Unlike the client side, the server side need not specify a particular username and password, instead it will accept any username and password.

- When a Web service requires additional security hardening, because, for example, they want to use a key that is different from the domain key generated for Oracle Fusion Applications, then you must use LPA and specify this key using a configuration override.

The Oracle Fusion Applications provisioning script generates a single keypair (public key and self signed certificate) with the alias orakey and stores the keypair in all Oracle Fusion Applications domains. All GPA policies will use this key by default unless you use LPA and specify a different key.

- When a Web service exists that must be invoked outside of an Oracle Fusion application that external service will be secured either with message protection or it must go through SSL to ensure protection. The default GPA for Oracle Fusion Applications is no message protection, which is not sufficient for external services that can be invoked outside Oracle Fusion applications. For external Web services, you must use LPA to use a message protection policy, for example to secure the external service to make it more secure or less secure.

- When a Fusion Web service can be invoked by an application outside of Oracle Fusion applications (because an Oracle Fusion application is integrated with the calling application) that service will most likely need to use policies to provide additional hardening. In this case, these clients should use LPA.

- Fusion Web service clients that need to connect to external non Fusion Web services will most likely need to use policies that are different from the globally attached policies. In this case, these clients should also use LPA.

You should use LPA whenever you want to override the globally attached policy. The user name and specifying an alternate key are common examples of overriding a globally attached policy.

In summary, use LPA when the Web service is a public service, when the service requires elevated privileges to connect using a particular user name and password, or when the service requires additional security hardening.

Related Links
The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For background about globally policy sets, see the “Understanding Oracle WSM Policy Framework” chapter of the Administering Web Services, in the Oracle Fusion Middleware Online Documentation Library.

### 53.3 Attach Policies Globally

All Oracle Fusion application Web services and Web service clients should use global policy attachment wherever possible. The developer and system administrator work together to enable GPA, as the following process describes.

To enable global policy attachment:
1. Application developer removes LPA from all clients and Web services (except for the situations that need to use LPA).

A system administrator can do this using either Oracle Enterprise Manager Fusion Middleware Control or using WebLogic Scripting Tool (WLST).

2. Application developer makes sure that the no_authentication_service_policy policy has been attached to those services that do not require authentication.

Note that unless you attach a no behavior policy (oracle/no_authentication_service_policy or oracle/no_authentication_client_policy), public Web services will inherit GPA and will no longer be accessible, thus it is necessary to perform this step before enabling GPA.

Developers can do this in Oracle JDeveloper directly in the ADF Business Components Web service implementation class file, as described in Attaching Policies Locally. Also, a system administrator can do this either using Oracle Enterprise Manager Fusion Middleware Control or using WebLogic Scripting Tool (WLST).

3. System administrator decides which policy to attach globally.

Profile choices are Authentication (AuthN), SSL and Message protection. Oracle Fusion Applications are configured to use the AuthN profile by default. For background about profiles choices, see Table 53-1.

4. System administrator creates the global policy sets.

A system administrator does this using either Oracle Enterprise Manager Fusion Middleware Control or using WLST. They will need to define a separate GPA policy set for each kind of service–SOA service, SOA reference, ADF Business Components Web service, and so on.

5. Application developer in cooperation with the system administrator verifies that the client and Web services are using GPA.

A system administrator does this using either Oracle Enterprise Manager Fusion Middleware Control or using WLST.

Related Links
The following documents provide additional information about Oracle Fusion Middleware topics discussed in this section:

- For details about how a system administrator creates global policy sets, see the "Managing Web Service Policies" chapter of the Administering Web Services, in the Oracle Fusion Middleware Online Documentation Library.

- For details about how a system administrator attaches local policies on Oracle WebLogic Server, see the "Attaching Policies to Web Services" chapter of the Administering Web Services, in the Oracle Fusion Middleware Online Documentation Library.

- For a summary of the predefined policies, see the "Predefined Policies" appendix of the Administering Web Services, in the Oracle Fusion Middleware Online Documentation Library.
• For details about how a system administrator creates global policy sets, see the "Creating and Managing Policy Sets" chapter of the *Administering Web Services*, in the Oracle Fusion Middleware Online Documentation Library.

## 53.4 Attaching Policies Locally

Because certain scenarios exist in which GPA cannot be used, Oracle Fusion application developers may need to use local policy attachment (LPA) for Web services and Web service clients. In some cases, LPA must be used on the service side and client side, while other cases exist where only the client side requires LPA.

You should use local policy attachment:

• When a Web service should be public (those that do not require user authentication)

• When a Web service client requires elevated privileges to connect using a particular user name and password (GPA only handles identity propagation, thus LPA is required when identity switching is required)

• When a Web service requires additional security hardening

• When Fusion Web service clients (for example, a JRF client or a SOA client) need to connect to external non Fusion Web services

The following table shows the recommended Oracle WSM policies and the components to which they apply.

### Table 53-1  Recommended Oracle Web Services Manager Policies for Oracle Fusion Applications

<table>
<thead>
<tr>
<th>Profile</th>
<th>Service Side Policy</th>
<th>Client Side Policy</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authentication</td>
<td>wss_saml_or_username_token_service_policy</td>
<td>Username: wss10_saml_token_client_policy</td>
<td>Performance: High</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Security: Low</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Authentication: password in clear, SAML token is unsigned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Wire level security: No</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Hardening: no</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Configuration: Easy. No key stores to set up.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Interoperability:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Username: High interoperates easily with many different stacks, including .NET, SOAP-UI, and more.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- SAML: Low. Unsigned SAML SV does not interoperate with most stacks.</td>
</tr>
</tbody>
</table>

Oracle Fusion Applications Developer's Guide
### Table 53-1 (Cont.) Recommended Oracle Web Services Manager Policies for Oracle Fusion Applications

<table>
<thead>
<tr>
<th>Profile</th>
<th>Service Side Policy</th>
<th>Client Side Policy</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SSL Profile</strong></td>
<td>wss_saml_or_username_token_over_ssl_service_policy</td>
<td>Username: oracle/wss_saml_or_username_token_over_ssl_service_policy</td>
<td>Performance: Medium</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Security: Medium</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Authentication: passwords encrypted because of 1-way-SSL, SAML token is signed by virtue of 2-way-SSL.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Wire level security: Transport level, using 1-way-SSL for username, and 2-way-SSL for SAML.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Hardening: Medium. Each application server can have its own separate key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Configuration: Hard. The same Oracle HTTP Server URI must be set up for both 1-way and 2-way-SSL and the client certificate must be set to propagate from Oracle HTTP Server to Oracle WebLogic Server. For details, see the &quot;Setting Up Your Environment for Policies&quot; chapter of the <em>Administering Web Services</em>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Interoperability:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Username: High</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- SAML: Medium</td>
</tr>
<tr>
<td><strong>Message Security</strong></td>
<td>wss11_saml_or_username_token_with_message_protection_service_policy</td>
<td>Username: wss11_username_token_with_message_protection_client_policy</td>
<td>Performance: Medium</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Security: High</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Wire level security: Transport level</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Hardening: High. Not only each server, but each Web service can have its own separate key.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Configuration: Medium. Key stores need to be set up.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Interoperability:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Username: Medium</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- SAML: Medium</td>
</tr>
<tr>
<td><strong>No Behavior</strong></td>
<td>oracle/no_authentication_service_policy</td>
<td>oracle/no_authentication_client_policy</td>
<td>No security. Service will be accessible.</td>
</tr>
</tbody>
</table>

### 53.4.1 How to Make a Web Service Publicly Accessible

One side effect of enabling GPA is that even public Web services (those that do not need any authentication) will now suddenly prompt for security credentials. To prevent this, all such Web services should use LPA to locally attach the `oracle/no_authentication_client_policy` on the client side and `oracle/no_authentication_service_policy` on the service side. After this is done,
these clients and Web services will ignore any global policy, and will work without authentication as before.

For example, suppose there is a non-Oracle external client calling a Fusion Web service. If this Web service did not have any security policy, and you turned on GPA, then this service will inherit the GPA setting, but because the client is an external client, it will not. Consequently, the service will be expecting secure messages that the client will not be sending, and the service will reject those messages.

In the case of an ADF Business Components Web service, you can enter service annotations in the Web service implementation class to specify the no behavior policy, as shown in the following example.

**Related Links**
The following documents provide additional information about Oracle Fusion Middleware topics discussed in this section:

- For more information about directly attaching the no behavior policy to a Web service endpoint, see the "Creating and Managing Policy Sets" chapter of the *Administering Web Services*, in the Oracle Fusion Middleware Online Documentation Library.

- For more information about ADF Business Components Web services, see the "Integrating Service-Enabled Application Modules" chapter of the *Developing Fusion Web Applications with Oracle Application Development Framework*, in the Oracle Fusion Middleware Online Documentation Library.

- For details about locally attaching the no behavior policy on Oracle WebLogic Server using Fusion Middleware Control and the WebLogic Scripting Tool (WLST), see the "Attaching Policies to Web Services" chapter of the *Administering Web Services*, in the Oracle Fusion Middleware Online Documentation Library.

**Example 53-1  Enabling No Behavior Policy for an ADF Business Components Web Service**

```java
@SecurityPolicy( { "oracle/no_authentication_service_policy"})
@CallbackSecurityPolicy("oracle/no_authentication_service_policy")
```

### 53.4.2 How to Support Elevated Privileges for Web Service Clients

By default, GPA supports identity propagation on the client side. However, because GPA is attached globally it is not possible to do any local configuration overrides. For example, if you have a Web service client that needs to connect using a particular user name and password (this is known as elevated privileges or identity escalation), then you cannot use GPA. With GPA you cannot specify this user name/password on a per client basis.

**Note:**

Even though you need to use LPA on the client side to perform configuration overrides, you can still use GPA on the service side. This is because on the service side you do not need a configuration override to set up a particular user name, instead you just attach the saml_or_username policy which will accept either user names (for identity escalation) or saml (for identity propagation).
Oracle Fusion Applications can use either RMI or SOAP to invoke the service. An RMI invocation of the service does not require security configuration. A SOAP invocation of the service can support identity propagation or identity switch.

To support identity propagation by the client, use the SAML token policy. To support identity switch, use the user name policy.

In the case of an ADF Business Components Web service, you can enter service annotations in the Web service implementation class to specify the locally attached policy, as shown in the following example.

Note that Fusion Web services should have asynchronous method calls enabled.

**Example 53-2 Attaching a Local Policy for an ADF Business Components Web Service**

```java
@SecurityPolicy("oracle/wss11_saml_or_username_token_with_message_protection_service_policy")
@CallbackSecurityPolicy("oracle/wss11_saml_token_with_message_protection_client_policy")
```

### 53.4.3 How to Provide Additional Security Hardening for Web Service Clients

Typically, Fusion Web services use a common domain wide key, which is necessary both for encryption and signing. Since this key is global it works very well with GPA. However, if certain Web services requires additional security hardening, because, for example, they want to use a key that is different from the domain key, then those services would need to use LPA and specify this key using a configuration override.

Another use case that would require LPA to provide additional security hardening exists when a non-Oracle Fusion application invokes an Oracle Fusion application Web service. In this case, because Fusion Web services do not use message protection by default, additional protection will be required to comply with the invoking application security policies.

**Related Links**

The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For details about locally attaching Web service client policy and configuring override properties on Oracle WebLogic Server, see the "Attaching Policies to Web Services" chapter in the *Administering Web Services*, in the Oracle Fusion Middleware Online Documentation Library.

### 53.4.4 How to Connect to Third Party Web Services

Fusion Web service clients that need to connect to external non-Fusion Web services will most likely need to use policies that are different from the globally set policy. Because the Oracle Fusion default for GPA is no message protection, this might not be sufficient for external services. In this case, clients should also use LPA.

**Related Links**

The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For details about locally attaching Web service client policy on Oracle WebLogic Server, see the "Attaching Policies to Web Services" chapter in the *Administering Web Services*, in the Oracle Fusion Middleware Online Documentation Library.
53.5 Authorize the Web Service with Entitlement Grants

When you want to secure an ADF Business Components Web service to require user authorization, you use JDeveloper to define entitlement-based function security policies directly in the file-based security repository for your Oracle Fusion application.

Before you begin:

You will need to complete the following tasks.

1. Consult a security administrator to export all predefined function security policies of the application that you are customizing into a `jazn-data.xml` file.

2. Copy the exported `jazn-data.xml` file into your application workspace.
   This is the file that JDeveloper will update when you create function security policies. In order for JDeveloper to use the file, copy the file to your application workspace in the `<JDevAppHome>/src/META-INF` folder.

To secure an ADF Business Components Web service:

1. In the exported `jazn-data.xml` file, grant access to the Web service using the JDeveloper security policy editor. For details, see How to Grant Access for the Service.

2. Enforce ADF Security authorization for the Web service in the Web service implementation class. For details, see How to Enforce Authorization for the Service.

Related Links

The following document provides additional information about Oracle Fusion Applications topics discussed in this section:

- For details about how the security administrator exports the application policy store, see the "Extracting Data from an OID Store to a File" section in the Administrator’s Guide.

53.5.1 How to Grant Access for the Service

Oracle ADF Security is responsible for authorizing Web services, that is, Oracle ADF Security decides whether a Web service is available to a given user by checking against the Oracle Platform Security Services (OPSS) policy store.

To grant access:

1. Create an entitlement to group one or more custom resources and their corresponding actions that together entitle end users to access the resource when needed to complete a specific duty.

   In the Oracle Fusion Applications environment, the basic security artifact for entitlement-based security policies is the entitlement (an entitlement is defined by an OPSS permission set).

2. Grant access to the Web service by defining an entitlement grant with a custom duty role that was added to the Oracle Fusion application policy store as the grantee.
The entitlement grant to the role specifies that the end user must be a member of the role to access the resources specified by the entitlement. As a security guideline, do not modify a predefined data security policy by granting additional permissions to a predefined duty role; you should use custom duty roles and you should not grant entitlements to predefined duty roles.

Example 53-3 Entitlement-Based Policy Definition in the jazn-data.xml File

```xml
<?xml version="1.0" ?>
<jazn-data>
   <policy-store>
      <applications>
         <application>
            <name>MyApp</name>
            <app-roles>
               <app-role>
                  <name>AppRole</name>
                  <display-name>AppRole display name</display-name>
                  <description>AppRole description</description>
                  <guid>F5494E409CFB11DEBFEBC112962B4F58</guid>
                  <class>oracle.security.jps.service.policystore.ApplicationRole</class>
               </app-role>
            </app-roles>
            <role-categories>
               <role-category>
                  <name>MyAppRoleCategory</name>
                  <display-name>MyAppRoleCategory display name</display-name>
                  <description>MyAppRoleCategory description</description>
               </role-category>
            </role-categories>
            <!-- resource-specific OPSS permission class definition -->
            <resource-types>
               <resource-type>
                  <name>WebserviceResourceType</name>
                  <display-name>WebserviceResourceType</display-name>
                  <description>Webservice Resource</description>
                  <matcher-class>oracle.wsm.security.WSFunctionPermission</matcher-class>
                  <actions-delimiter>,</actions-delimiter>
                  <actions>invoke</actions>
               </resource-type>
            </resource-types>
            <resources>
               <resource>
               </resource>
               <resource>
                  <name>http://xmlns.oracle.com/apps/contracts/termsAuthoring/deliverables/service/DeliverableService#findDeliverableByDeliverableId</name>
               </resource>
            </resources>
         </application>
      </applications>
   </policy-store>
</jazn-data>
```
<permission-sets>
  <permission-set>
    <name>MyWebServiceEntitlement</name>
    <display-name>MyEntitlement display name</display-name>
    <description>MyEntitlement description</description>
    <member-resources>
      <member-resource>
        <type-name-ref>WebserviceResourceType</type-name-ref>
        <resource-name>xmlns.oracle.com/apps/financials/subledgerAccounting
          /accountingMethodSetup/accountRulesService/AccountRulesService#*
        </resource-name>
        <actions>invoke</actions>
      </member-resource>
      <member-resource>
        <type-name-ref>WebserviceResourceType</type-name-ref>
        <resource-name>http://xmlns.oracle.com/apps/contracts/
termsAuthoring/deliverables/service/
  DeliverableService#findDeliverableByDeliverableId
</resource-name>
        <actions>invoke</actions>
      </member-resource>
    </member-resources>
  </permission-set>
</permission-sets>

<!-- Oracle function security policies -->
<jazn-policy>
  <!-- function security policy is a grantee and permission set -->
  <grant>
    <!-- application role is the recipient of the privileges -->
    <grantee>
      <principals>
        <principal>
          <class>
            oracle.security.jps.service.policystore.ApplicationRole
          </class>
          <name>AppRole</name>
          <guid>F5494E409CFB11DEBFEBC11296284F58</guid>
        </principal>
      </principals>
    </grantee>

    <!-- entitlement granted to an application role -->
    <permission-set-refs>
      <permission-set-ref>
        <name>MyWebServiceEntitlement</name>
      </permission-set-ref>
    </permission-set-refs>
  </grant>
</jazn-policy>
</application>
</applications>
</jazn-data>

For details about creating entitlement-based security policies using JDeveloper tools, see How to Create Entitlement Grants for Custom Application Roles.

The following example shows a complete set of required grants enabling Web service authorization. The entitlement definition MyWebServiceEntitlement defines an
invoke action permission for two Web service member resources, where the resource identified as `AccountRulesService#*` uses the asterisk wildcard character to identify a set of resources (where the asterisk specifies any number of arbitrary characters). The `jazn-policy` definition grants the invoke permission conferred by `MyWebServiceEntitlement` to the members of the `AppRole`.

**Related Links**

The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For details about the ADF Security, see the "Enabling ADF Security in a Fusion Web Application" chapter in the *Developing Fusion Web Applications with Oracle Application Development Framework*, in the Oracle Fusion Middleware Online Documentation Library.

### 53.5.2 How to Enforce Authorization for the Service

In the case of ADF Business Components Web services, there is no need to run the ADF Security wizard to enforce authorization checking on the defined Web service security policies. Additionally, Oracle Fusion applications do not use Oracle WSM policies for authorization. Instead, the Oracle Fusion application uses the policy interceptor defined by the ADF Business Components to enforce authorization checking. The service factory is used to invoke an ADF Business Components service synchronously within a domain. Security information is passed from the calling program to the service automatically. And, whether the service is invoked by the service factory directly or through a BPEL process, authorization is enforced by the EJB implementation of the ADF Business Components Web service.

The following table shows the recommended policy interceptor used to enforce entitlement-based policies for Oracle Fusion applications and the components to which they apply.

<table>
<thead>
<tr>
<th>On this component...</th>
<th>Use this interceptor...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADF Business Components Web services (at the service or operation level)</td>
<td>Use <code>ServicePermissionCheckInterceptor</code> to implement authorization.</td>
</tr>
</tbody>
</table>

The ADF Business Components policy interceptor works for both RMI and SOAP cases and supports the EJB implementation of ADF Business Components Web services. Therefore, if `ServicePermissionCheckInterceptor` is specified in the ADF Business Components Web service implementation class, an Oracle WSM authorization policy is not required for Fusion Web services.

In the case of an ADF Business Components Web service, you enter the `@Interceptors` annotation and import statements in the Web service implementation class to specify the policy interceptor, as shown in the following example.

In order for this interceptor to work, you need to configure the policy interceptor in your `ejb-jar.xml` file. In the Application Navigator, expand the `META-INF` node of the Web service project and double-click the `ejb-jar.xml` node. In the source editor, add the `JpsInterceptor` definition required by the EJB for authorization checking, as shown in the following example.
Related Links
The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For details about the ADF Business Components ServiceFactory class, see the "Integrating Service-Enabled Application Modules" chapter in the Developing Fusion Web Applications with Oracle Application Development Framework, in the Oracle Fusion Middleware Online Documentation Library.

Example 53-4  Enforcing Authorization with ADF Business Components Policy Interceptor

```java
import oracle.jbo.server.svc.ServicePermissionCheckInterceptor;
import oracle.jbo.server.svc.ServiceContextInterceptor;
@Interceptors({ServiceContextInterceptor.class, ServicePermissionCheckInterceptor.class})
public class xxxServiceImpl ...
```

Example 53-5  Configuring the JPSInterceptor for the Application in the ejb-jar.xml File

```xml
<ejb-jar xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/j2ee/ejb-jar_3_0.xsd" version="3.0"
xmnl="http://java.sun.com/xml/ns/javaee">
<enterprise-beans>
...
</enterprise-beans>
<interceptors>
<interceptor>
<interceptor-class>
oracle.security.jps.ee.ejb.JpsInterceptor
</interceptor-class>
<env-entry>
<env-entry-name>application.name</env-entry-name>
<env-entry-type>java.lang.String</env-entry-type>
<env-entry-value>ApplicationName</env-entry-value>
<injection-target>
<injection-target-class>
oracle.security.jps.ee.ejb.JpsInterceptor
</injection-target-class>
<injection-target-name>
application_name
</injection-target-name>
</injection-target>
</env-entry>
</interceptor>
...
</interceptors>
<assembly-descriptor>
<interceptor-binding>
<ejb-name>*</ejb-name>
<interceptor-class>
oracle.security.jps.ee.ejb.JpsInterceptor
</interceptor-class>
</interceptor-binding>
</assembly-descriptor>
</ejb-jar>
```
53.6 What Happens At Runtime: How Policies Are Enforced

When a service is invoked there is security for the client side (caller) and for the server side (callee).

The client side can invoke the service through a SOAP service invocation, where the client can be a JAX-WS proxy or a SOA composite (service factory via SOAP is still JAX-WS proxy). Invoking the service, requires the client to pass the security context to the server side; the client side will either propagate the current user credential or switch identity, based on the client side authentication policy.

Alternatively, the client side can invoke the service through an RMI invocation. This type of invocation applies only to the ADF Business Component Web service. In this case, there is no client side authentication policy, since the client side always just propagates the identity to the server side.

Security for the server side is based on the Oracle WSM authentication policy. The server side first authenticates the user. A SOA composite service will not perform authorization, but ADF Business Components Web services will check whether the user is authorized to invoke the service. This type of service performs the authorization check using the EJB `ServicePermissionCheckInterceptor` interceptor before executing any service method.

The following figure illustrates how Oracle WSM policies are enforced within the Oracle Fusion Applications use case described in Introduction to Securing Web Services Use Cases.

**Figure 53-2 Using Oracle Web Services Manager Security Policies**

---

Security Policy Legend

- **Client**: Client side
- **A1**: Authentication
- **A2**: Authorization
This example use case illustrates the following:

- Each SOAP client includes an attached client policy.
- Authorization is enforced in the ADF Business Components Web services only.
- SOA components have no authorization policies.
- Oracle Web Services Manager policies do not apply to events.

Related Links

The following document provides additional information about Oracle Fusion Middleware topics discussed in this section:

- For more information about Oracle Web Services Manager policies, see the "Understanding Oracle WSM Policy Framework" chapter of the Administering Web Services, in the Oracle Fusion Middleware Online Documentation Library.

## 53.7 Maintain Application Session Context Across Web Service Requests

When the user invokes a function defined by a Web service, the current application session context must be propagated across Web service requests. The application session context contains important information that is stored for the duration of the user's HTTP session. This includes information that defines a context for the application, such as its language preferences, date, and number formatting.

In order for application session context propagation to occur, you need to register the 
App1SessionContext class with the context interceptor infrastructure. Then when a SOAP request is generated by the invoked Web service, the request calls out to the infrastructure and adds the application session context onto the SOAP payload.

To register the ApplSessionContext class, you must add the oracle.applcore.config library to the weblogic-application.xml file for projects on both ends of the Web service request. In order for propagation to work, both the client sending the request and the server receiving the request must have this library.

Before you begin:

It may be helpful to have an understanding of application user sessions. For more information, see Implementing Application User Sessions.

You will need to complete this task:

- Configure your project to use application user sessions. For more information, see Configuring Your Project to Use Application User Sessions.

To add the Oracle Applications Core (Config) JDev library to the classpath:

1. In the Application Resources panel of the Application Navigator, double-click the weblogic-application.xml file of the project that defines the incoming side of the Web service request.

2. In the source editor for the file, add the following lines alongside the other <library-ref> tags:

```xml
<library-ref>
  <library-name>
    oracle.applcore.config
  </library-name>
</library-ref>
```
The order of the `<library-ref>` tags is not important; however, all `<library-ref>` tags must be grouped together.

3. Repeat this change in the project that defines the outgoing side of the Web service request.

In order for propagation to work, both the client sending the request and the server receiving the request must have the `oracle.applcore.config` library.
Secure End-to-End Portlet Applications

This chapter describes how to authenticate and authorize portlet services, as well as configure key and credential stores. The process of securing portlet services is similar to that of securing web services.

- Introduction to Securing End-to-End Portlet Applications
- Securing the Portlet Service
- Securing the Portlet Client
- Registering the Key Store and Writing to the Credential Store

54.1 Introduction to Securing End-to-End Portlet Applications

In Oracle Fusion Applications, portlets are WSRP portlets, therefore, web services. Oracle Web Services Manager (WSM) policies secure portlets, in the same way that they secure ordinary web services.

Oracle Web Services Manager implements web service security, and allows for run time enforcement and declarative policy attachment within Oracle Fusion Middleware.

Oracle Fusion applications make use of an Oracle WSM feature called global policy attachment (GPA). In GPA, policies are not attached locally, but specified at a global level. At runtime, components inherit the global policy and Oracle WSM enforces it.

For each portlet, these four ports require Oracle WSM policies:

- WSRP_v2_Markup_Service
- WSRP_v2_PortletManagement_Service
- WSRP_v2_Registration_Service
- WSRP_v2_ServiceDescription_Service

Only the WSRP_v2_Markup_Service markup port requires an authentication policy. By default, no policy should be locally attached to the markup port; this port will inherit the policy from GPA.

However, if the WSRP_v2_Markup_Service port has unique requirements not fulfilled by GPA, then a locally attached policy will be necessary. Additionally, if the locally attached policy specifies message protection or SSL, the necessary key store setup must be in place.

The three non-markup ports are anonymous and therefore you will need to locally attach a "no behavior" policy (defined by oracle/no_authentication_service_policy) to override GPA.
The requirements for the client counterparts for each of these ports is the same. Clients of the WSRP_v2_Markup_Service port inherit GPA, and clients of the three non-markup ports propagate an anonymous token defined by the oracle/no_authentication_client_policy policy.

The following table summarizes the policies attached to the portlet service and the client.

<table>
<thead>
<tr>
<th>Port</th>
<th>Service Side Policy</th>
<th>Client Side Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSRP_v2_Markup_Service</td>
<td>No local policy. Inherits from GPA, which by default is oracle/wss_saml_or_username_token_service_policy.</td>
<td>No local policy. Inherits from GPA, which by default is oracle/wss_saml_or_username_token_client_policy.</td>
</tr>
<tr>
<td>WSRP_v2_PortletManagement_Service</td>
<td>Local anonymous policy: oracle/no_authentication_service_policy</td>
<td>Local anonymous policy: oracle/no_authentication_client_policy</td>
</tr>
<tr>
<td>WSRP_v2_Registration_Service</td>
<td>Local anonymous policy: oracle/no_authentication_service_policy</td>
<td>Local anonymous policy: oracle/no_authentication_client_policy</td>
</tr>
<tr>
<td>WSRP_v2_ServiceDescription_Service</td>
<td>Local anonymous policy: oracle/no_authentication_service_policy</td>
<td>Local anonymous policy: oracle/no_authentication_client_policy</td>
</tr>
</tbody>
</table>

To override GPA and secure end-to-end portlet applications with a locally attached policy:

- Secure the portlet service.
- Secure the portlet client.
- Register the key store.
- Write to the credential store.

### 54.2 Secure the Portlet Service

Securing the portlet service with a locally attached policy that overrides GPA involves the following main steps:

- Authenticating the service
- Configuring the key store and credential store
- Authorizing the service

### 54.2.1 How to Authenticate the Service

Authenticating the service is necessary only in two cases:
When applying the "no behavior" policy to the anonymous ports.

When GPA is being overridden for the WSRP_v2_Markup_Service port.

When a policy is attached locally, Oracle ADF must authenticate the portlet service against an Oracle Web Services Manager policy, such as wss10_saml_token_with_message_protection_service_policy. In addition, it is necessary to configure security for the Oracle Fusion web application EAR file.

Authenticating the services involves the following main steps:

- Attach the policy (for example, wss10_saml_token_with_message_protection_service_policy) to the provider. You can do this in one of the following ways:
  - Use Oracle Enterprise Manager.
  - Alternatively, manually update oracle-webservices.xml. This is a packaging artifact, meaning it is not available in Oracle JDeveloper during design time. To edit the file, deploy your application to an EAR file. Extract the oracle-webservices.xml file, update it and repackage it into the EAR file.

To edit the oracle-webservices.xml file when overriding GPA:

Open the oracle-webservices.xml file, find port-component name="WSRP_v2_Markup_Service" and add the code shown in the following example.

Example 54-1 Edit the oracle-webservices.xml File

```xml
<port-component name="WSRP_v2_Markup_Service" style="document"
    binding QName="(urn:oasis:names:tc:wsrp:v2:bind)WSRP_v2_Markup_Binding_SOAP" enabled="true"
    schemaValidateInput="false">
  <policy-references>
    <policy-reference enabled="true" uri="oracle/wss10_saml_token_with_message_protection_service_policy" category="security"/>
  </policy-references>
<brick operations>
  <operation name="performBlockingInteraction" inputName="performBlockingInteraction"
    outputName="performBlockingInteractionResponse"
    input="(urn:oasis:names:tc:wsrp:v2:types)performBlockingInteraction" use="literal"/>
  <operation name="releaseSessions" inputName="releaseSessions"
    outputName="releaseSessionsResponse"
    input="(urn:oasis:names:tc:wsrp:v2:types)releaseSessions" use="literal"/>
  <operation name="getMarkup" inputName="getMarkup" outputName="getMarkupResponse"
    input="(urn:oasis:names:tc:wsrp:v2:types)getMarkup" use="literal"/>
  <operation name="handleEvents" inputName="handleEvents" outputName="handleEventsResponse"
    input="(urn:oasis:names:tc:wsrp:v2:types)handleEvents" use="literal"/>
  <operation name="initCookie" inputName="initCookie" outputName="initCookieResponse"
    input="(urn:oasis:names:tc:wsrp:v2:types)initCookie" use="literal"/>
  <operation name="getResource" inputName="getResource" outputName="getResourceResponse"
    input="(urn:oasis:names:tc:wsrp:v2:types)getResource" use="literal"/>
</brick operations>
</port-component>
<!-- start:deployment time generated info -->
<deployment>
  <tie-class-name>oasis.names.tc.wsrp.v2.bind.runtime.WSRP_v2_Markup_Binding_SOAP_Tie</tie-class-name>
  <service QName namespaceURI="urn:oasis:names:tc:wsrp:v2:wadl" localpart="WSRP_v2_Service"/>
  <soap-version>soap1.1</soap-version>
</deployment>
<!-- end:deployment time generated info -->

Secure End-to-End Portlet Applications 54-3
54.2.2 How to Configure the Key Store and Credential Store

By default, a globally attached policy profile specifies (authentication [AuthN]) and there is no need to use Oracle Enterprise Manager to configure a key store or a credential store. You only need to perform this task when a policy has a message protection or a SSL profile.

The key store contains the signing and encryption keys used to encrypt and decrypt messages. The key store itself and all the keys are password protected. The keys are also referred to using aliases, which are stored, along with their corresponding passwords, in the credential store. When accessing the key store, query the credential store first for the necessary aliases and passwords.

You can verify the creation of the key store and credential store as follows.

To verify the creation of the key store and credential store:

1. Open $DOMAIN_HOME/config/fmwconfig/jps-config.xml.
2. Verify the existence of the entry shown in the following example. This code should be configured out of the box.
3. Make sure the default context references the key store and credential store service instances as shown in the following example.

Note:

There is no need to restart your domain if this configuration is already in place.

Example 54-2 Verify the credstore and keystore serviceInstance Elements

Example 54-3 Default Context References to the Credential Store and Key Store
54.2.3 How to Authorize the Service

Oracle ADF Security is responsible for authorizing portlets, that is, Oracle ADF Security decides whether a portlet is available to a given user by checking it against the Oracle Platform Security Services (OPSS) policy store. Portlets are just one way of exposing local task flows to remote applications. A component called a portlet bridge is responsible for bridging between portlets and task flows. A portlet bridge enables exposing a task flow as a portlet.

After you create an entitlement grant for the desired task flow in jazn-data.xml, you must create a resource grant for the portlet bridge component to the authenticated role, as shown in the following example.

The following examples shows an entitlement grant enabling access to the task flow.

Note when the Oracle Fusion application needs to provide anonymous access to a portlet, the bridge wrapper task flow needs a grant to the anonymous-role, and the markup port needs a no_authentication policy, or it can use GPA, but needs to specify a Default User in the producer registration, using a valid guest user account, as described in Securing the Portlet Client.

Example 54-4  Resource Grant to the Authenticated Role

<jazn-policy>
  <grant>
  <grantee>
    <principals>
      <principal>
        <class>oracle.security.jps.internal.core.principals.JpsAuthenticatedRoleImpl</class>
        <name>authenticated-role</name>
      </principal>
    </principals>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.adf.controller.security.TaskFlowPermission</class>
      <name>/WEB-INF/adfp-portlet-bridge-container.xml#adfp-portlet-bridge-container</name>
      <actions>view</actions>
    </permission>
  </permissions>
  </grant>
  ...
</jazn-policy>

Example 54-5  Entitlement-Based Policy Definition in the jazn-data.xml File

<?xml version="1.0" ?>
<jazn-data>
  <policy-store>
    <applications>
      <application>
        <name>MyApp</name>
        <app-roles>
          <app-role>
            <name>AppRole</name>
            <display-name>AppRole display name</display-name>
            <description>AppRole description</description>
            <guid>F5494E409CFCB11DEBF6FC11296284F58</guid>
          </app-role>
        </app-roles>
      </application>
    </applications>
  </policy-store>
</jazn-data>
<class>oracle.security.jps.service.policystore.ApplicationRole</class>
</app-role>
</app-roles>

<!-- resource-specific OPSS permission class definition -->
<resource-types>
  <resource-type>
    <name>TaskFlowResourceType</name>
    <display-name>Task Flow</display-name>
    <description>Task Flow resource type</description>
    <matcher-class>oracle.adf.controller.security.TaskFlowPermission</matcher-class>
    <actions-delimiter>,</actions-delimiter>
    <actions>view,customize,grant,personalize</actions>
  </resource-type>
</resource-types>

<resources>
  <resource>
    <name>/WEB-INF/my-task-flow.xml#my-task-flow</name>
    <display-name>my-task-flow</display-name>
    <description>/WEB-INF/my-task-flow</description>
    <type-name-ref>TaskFlowResourceType</type-name-ref>
  </resource>
</resources>

<!-- entitlement definition -->
<permission-sets>
  <permission-set>
    <name>MyPortletEntitlement</name>
    <member-resources>
      <member-resource>
        <resource-name>/WEB-INF/my-task-flow.xml#my-task-flow</resource-name>
        <type-name-ref>TaskFlowResourceType</type-name-ref>
        <display-name>my-task-flow</display-name>
        <actions>view</actions>
      </member-resource>
    </member-resources>
  </permission-set>
</permission-sets>

<!-- Oracle function security policies -->
<jazn-policy>
  <!-- function security policy is a grantee and permission set -->
  <grant>
    <!-- application role is the recipient of the privileges -->
    <grantee>
      <principal>
        <class>
          oracle.security.jps.service.policystore.ApplicationRole
        </class>
        <name>AppRole</name>
      </principal>
    </grantee>
  </grant>
  <!-- entitlement granted to an application role -->
  <permission-set-ref>
    <permission-set-ref>
      <name>MyPortletEntitlement</name>
      <member-resources>
        <member-resource>
          <resource-name>/WEB-INF/my-task-flow.xml#my-task-flow</resource-name>
          <type-name-ref>TaskFlowResourceType</type-name-ref>
          <display-name>my-task-flow</display-name>
          <actions>view</actions>
        </member-resource>
      </member-resources>
    </permission-set-ref>
  </permission-set-ref>
</jazn-policy>
54.3 Secure the Portlet Client

Securing the portlet client is necessary only when applying the "no behavior" policy to the anonymous ports.

Securing a portlet consumer, or client, means enabling identity propagation. You can enable identity propagation while registering the portlet producer in Oracle JDeveloper. When registering the portlet producer, make sure you select following values in the WSRP Portlet Producer Registration wizard.

In the Configure Security Attributes window, select the following:

- **Token Profile**: Select a policy that overrides GPA. For example, you might select No Authentication Client Policy when locally attaching a policy for any of the three non-markup ports.

- **Configuration**: Select Default.

- **Default User**: Leave empty or enter a valid guest user account ID to propagate to the portlet.

  Note that **Default User** is only used when the consumer identity is in fact anonymous. In this case, the **Default User** field lets you specify some valid identity that is necessary to propagate to the portlet, when the consumer is anonymous, and the producer needs to receive a valid identity.

Within the portlet consumer domain, make sure the key store and credential store are the same ones used by the portlet producer or service. The key store and credential store are located at $DOMAIN_HOME/config/fmwconfig. For more information, see the section about verifying the creation of the key store and credential store under How to Configure the Key Store and Credential Store.

54.4 Register the Key Store and Writing to the Credential Store

By default, globally attached policy profile specifies (authentication [AuthN]) and there is no need to use Oracle Enterprise Manager to register a key store or create a credential store. You only need to perform this task when a policy profile offers message protection or SSL.

However, if you need to configure another key store for your domain, use Oracle Enterprise Manager to register the key store and write to the credential store.

54.4.1 How to Register the Key Store and Write to the Credential Store

To register the key store and write to the credential store:

1. In Oracle Enterprise Manager, expand your domain. Select **WebLogic Domain > WebLogic Domain Name**.
2. In the right-hand pane, click the WebLogic Domain menu at the top of the page and select Security > Credentials.

3. If there is no map called oracle.wsm.security, create one. If the map exists, skip this step. The following figure displays the Create Map window.

\[\text{Figure 54-1  Create a Map If None Exists}\]

- Click the Create Map button.
- For the Map Name, enter oracle.wsm.security.
  
  Do not create any keys. Keys are created when configuring the domain's service provider.

4. In the right-hand pane, click the WebLogic Domain menu at the top of the page and select Security > Security Provider Configuration.

5. In the Service Provider Configuration page, under the Key Store section, click the Configure button. The Service Provider Configuration page is shown in the following figure.

\[\text{Figure 54-8  Register the Key Store and Writing to the Credential Store}\]
6. If the Keystore Path displays .default-keystore.jks, follow the instructions here. Otherwise, skip this step.

Uncheck the **Configure KeyStore Management** box and click **OK**. This displays the window shown in the previous figure.

Under the Keystore section, click **Configure** again and enter the following information. The file producer.jks is assumed to be located under the directory path $DOMAIN_HOME/config/fmwconfig which contains a certificate alias called producer. The following figure displays the Keystore Configuration page.
Figure 54-3  Configure the Keystore

- **Keystore Path**: Enter the path of the keystore, in this case ./producer.jks.
- **Password/Confirm Password**: Enter the required password, then confirm the password.

**Signature Key**
- **Key Alias**: Enter the name of the signature key.
- **Signature Password/Confirm Password**: Enter the required password, then confirm the password.

**Encryption Key**
- **Crypt Alias**: Enter the name of the encryption key.
- **Crypt Password/Confirm Password**: Enter the required password, then confirm the password.

7. Restart your domain.

### 54.4.2 What Happens When You Register the Key Store and Write to the Credential Store

Entering this information enables the creation of keystore-csf-key, sign-csf-key and enc-csf-key in the credential store of the domain. You can verify that the keys have been created by viewing the credential store page of the domain in Oracle Enterprise Manager.

### 54.5 Maintain Application Session Context Across Web Service Requests

When the user invokes a function defined by a web service, the current application session context must be propagated across web service requests. The application session context contains important information that is stored for the duration of the user’s HTTP session. This includes information that defines a context for the application, such as its language preferences, date, and number formatting.

In order for application session context propagation to occur, you need to register the ApplSessionContext class with the context interceptor infrastructure. Then when a
SOAP request is generated by the invoked web service, the request calls out to the infrastructure and adds the application session context onto the SOAP payload.

To register the ApplSessionContext class, you must add the oracle.applcore.config library to the weblogic-application.xml file for projects on both ends of the web service request. In order for propagation to work, both the portlet consumer and the portlet producer must have this library.

**Before you begin:**

It may be helpful to have an understanding of application user sessions. For more information, see Implementing Application User Sessions.

You will need to complete this task:

Configure your project to use application user sessions. For more information, see Configuring Your Project to Use Application User Sessions.

**To add the Oracle Applications Core (Config) JDev library to the classpath:**

1. In the Application Resources panel of the Application Navigator, double-click the weblogic-application.xml file of the project that defines the portlet producer.

2. In the source editor for the file, add the following lines alongside the other <library-ref> tags:

   ```xml
   <library-ref>
     <library-name>
       oracle.applcore.config
     </library-name>
   </library-ref>
   
   The order of the <library-ref> tags is not important; however, all <library-ref> tags must be grouped together.

3. Repeat this change in the project that defines the portlet consumer.

   In order for propagation to work, both the portlet consumer and the portlet producer must have the oracle.applcore.config library.
This chapter provides a basic overview of the Oracle WebLogic Server environment and information about how to run your applications on Integrated WebLogic Server. It also provides information about how to deploy your applications to the Administration Server instance of WebLogic Server to perform end-to-end testing of new applications. If you are deploying customizations or extensions, see the Oracle Fusion Applications Extensibility Guide for Developers.

This chapter includes the following sections:

- Introduction to Deploying Applications to Oracle WebLogic Server
- Running Applications on Integrated WebLogic Server
- Preparing to Deploy Oracle ADF Applications to an Instance of Standalone WebLogic Server
- Deploying Your Oracle ADF Applications to an Administration Server Instance of WebLogic Server
- Deploying Your SOA Projects to an Administration Server Instance of WebLogic Server

The scope of this chapter is limited to what is unique in the Oracle Fusion Applications environment. For general details about Oracle WebLogic Server, references are made to the generic Oracle Fusion Middleware guides.

55.1 Introduction to Deploying Applications to Oracle WebLogic Server

Deployment is the process of packaging application files as an archive file and installing it to a target application server. You can use JDeveloper to deploy your Oracle Applications Development Framework (Oracle ADF) applications or SOA applications directly to Oracle WebLogic Server or indirectly to an archive file as the deployment target, and then install this archive file to the target server. You also can run Oracle ADF applications (but not SOA applications) in JDeveloper using Integrated WebLogic Server.

JDeveloper contains Integrated WebLogic Server that allows the developer to run the application using the Run command. You do not need to manually create deployment descriptors or WebLogic Server domains, as you would if you were deploying the application to an instance of Standalone WebLogic Server. However, Standalone WebLogic Server allows more configuration options, such as support for SOA. For more information on using Integrated WebLogic Server, see Running Applications on Integrated WebLogic Server.

If you are deploying the application to a standalone WebLogic Server instance, you may need to perform security-related configuration to prepare the application for
deployment. You may need to create or edit deployment descriptors and deployment profiles to prepare the application.

Whether you are using standalone or Integrated WebLogic servers to host the application, you will need to configure the WebLogic domains for Oracle Fusion Applications. You must run the Configure Fusion Domain Wizard from JDeveloper to configure the Integrated WebLogic Server or create a property file that is used to configure standalone WebLogic Server instances. For more information about the wizard, see Setting Up Your Development Environment.

Table 55-1 describes some common deployment techniques that you can use during the application development and deployment cycle.

<table>
<thead>
<tr>
<th>Deployment Technique</th>
<th>Environment</th>
<th>When to Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run directly from JDeveloper</td>
<td>Test or Development</td>
<td>When you are developing your application. You want deployment to be quick because you will be repeating the editing and deploying process many times. JDeveloper contains Integrated WebLogic Server, on which you can run and test your application.</td>
</tr>
<tr>
<td>Use JDeveloper to directly deploy to the target application server</td>
<td>Test or Development</td>
<td>When you are ready to deploy and test your application on an application server in a test environment. This also can be done on Integrated WebLogic Server, and it may be required in cases where there may be nothing to Run, such as deploying an ADFbc service. For example, you can also use the test environment to develop your deployment scripts.</td>
</tr>
<tr>
<td>Use JDeveloper to deploy to an EAR file, then use the Oracle WebLogic Server Administration console, WLST commands, or Enterprise Manager for deployment.</td>
<td>Test or Development</td>
<td>When you are ready to deploy and test your application on an application server in a test environment. As an alternative to deploying directly from JDeveloper, you can deploy to an EAR file and then use other tools to deploy to the WebLogic Server instance. This also can be done on Integrated WebLogic Server. You can also use the test environment to develop your deployment scripts.</td>
</tr>
</tbody>
</table>

After you have performed the tasks required for standalone deployment, you can use JDeveloper to deploy directly to a WebLogic Server instance or to create an Enterprise Archive (EAR) file and deploy the EAR file using WebLogic Server Administration Console, Enterprise Manager, or WebLogic Scripting Tool (WLST) commands.

For more information on how to deploy an application directly using JDeveloper, see Deploying Your Oracle ADF Applications to an Administration Server Instance of WebLogic Server.

For more information about deploying the application using WebLogic Server Administration Console, or WLST, see the Oracle Fusion Middleware Administering Oracle Fusion Middleware guide and the Oracle Fusion Middleware Administering Oracle ADF Applications guide.
Note:
This chapter discusses deploying applications. If you are deploying customizations or extensions, see the “Deploying Oracle ADF Customizations and Extensions” and the “Deploying SOA Composite Applications Customizations and Extensions” sections of the Extensibility Guide for Developers.

55.1.1 Prerequisites for Deployment

Before you deploy an application, you should perform the following tasks:

1. You must run the Configure Fusion Domain Wizard to set up the Integrated WebLogic Server domain or create a property file to be used to set up the standalone WebLogic Server domain. For instructions, see Setting Up Your Development Environment.

2. If you want more debugging output, set the environment variable using one of the shell commands as shown in Example 55-1, before starting the Administration Server instance of WebLogic Server or before starting JDeveloper if you are going to run Integrated WebLogic Server.

   Note: If using Integrated WebLogic Server, you might want to set this on run-time properties so that this can be controlled directly from JDeveloper.

3. You must disable the Auto Generate and Synchronize weblogic-jdbc.xml Descriptors During Deployment option to use the Global Java Database Connectivity (JDBC) datasource. To do this, open Application Properties and choose the Deployment category as shown in Figure 55-1. Unselect the Auto Generate and Synchronize weblogic-jdbc.xml Descriptors During Deployment option and click OK.

   For additional details, see Setting Up JDBC Data Sources on Oracle WebLogic Server.
Example 55-1 Setting Environment Variable

# For csh shell
setenv JAVA_OPTIONS "$JAVA_OPTIONS -Djbo.debugoutput=console"

# For bash shell
export JAVA_OPTIONS="$JAVA_OPTIONS -Djbo.debugoutput=console"

For Windows
set JAVA_OPTIONS=%JAVA_OPTIONS% -Djbo.debugoutput=console

55.1.1.1 Setting Up JDBC Data Sources on Oracle WebLogic Server

To avoid passwords being present in plain text in deployed files, JDeveloper uses password indirection, which means that passwords for the data sources must be set on the server before the application will run correctly.

You do this using global data sources, which are set up in the Oracle WebLogic Server Administration Console using the Data Sources link under JDBC.

JDeveloper ensures that your web application web.xml, or EJB application ejb-jar.xml, contains the necessary <resource-ref> entry to identify an application resource name. The name is jdbc/connection-nameDS, where connection-name is the name of the application resources connection.

The application looks up this data source using the application-specific resource JNDI namespace of java:comp/env/jdbc/connection-nameDS, and it finds this resource because web.xml contains the <resource-ref> entry for jdbc/connection-nameDS.

An important control for the files that are generated is the Auto Generate and Synchronize weblogic-jdbc.xml Descriptors During Deployment field on the WebLogic page of the Application Properties dialog.

When the Auto Generate field is selected, JDeveloper does the following:
• Generates an application-name-jdbc.xml file for each connection in the application resources, and sets the indirect password attribute.

```xml
<jdbc-driver-params>
  <use-password-indirection>true</use-password-indirection>
</jdbc-driver-params>
```

Upon deployment, JDeveloper determines the JDBC connection password from the username in application-name-jdbc.xml, and populates the JDBC connection password using an Mbean.

• `weblogic-application.xml` is updated to add each application-name-jdbc.xml as a module.

• `web.xml` (if it exists) has a resource reference to each JDBC JNDI name.

### 55.1.1.1 How to Create a Global Data Source on Oracle WebLogic Server

You create a global data source on Oracle WebLogic Server Administration Console. To set up a global data source:

1. Login to the Oracle WebLogic Server Administration Console.
2. Click the **Data Sources** link under **JDBC**.
3. On the Summary of JDBC Data Sources page, click **New**.
4. In the Create a New JDBC Data Source page, enter details of the data source.
   - The name can be anything.
   - The JNDI name must be of the form `jdbc/connection-nameDS`. For example, if the application has a connection name `connection1`, the JNDI name is `jdbc/ connection1DS`.
5. Ensure that the database type is Oracle and that the driver is Oracle's Driver (Thin) for Service Connections; Version 9.0.1, 9.2.0, 10, 11.
6. Click **Next** twice to navigate to the Create a New JDBC Data Source page, where you enter the connection details.
   - The database name is the Oracle SID.
   - The host name is the name of the machine the database is on.
   - The default port is 1521.
7. Enter the user name and password, for example `hr/hr`.
8. Click **Next** and click **Test Configuration**.
9. Click **Next** to navigate to the Select Targets page, where you select a target for this data source. If you fail to select a target, the data source is created but not deployed.
10. Click **Finish**.

### 55.1.1.2 Deploy to an EAR File to Run on Oracle WebLogic Server

To deploy an application to an EAR file to run on Oracle WebLogic Server, you can:
• Select the **Auto Generate and Synchronize weblogic-jdbc.xml Descriptors During Deployment** field, and set up passwords using application level credential mapping.

• Alternatively, you can deselect the **Auto Generate and Synchronize weblogic-jdbc.xml Descriptors During Deployment** field and set up passwords by creating a global data source on Oracle WebLogic Server.

If you are deploying using ojdeploy:

• You can use the **-nodatasources** switch, in which case you can set up passwords on Oracle WebLogic Server by either:
  
  – Creating a global data source.
  
  – Manually creating application level data sources.

• If you do not use the **-nodatasources** switch, you can only set up passwords using application level credential mapping.

### 55.1.2 Introduction to the Standalone Administration Server WebLogic Server Instance

A Weblogic Server instance is a configured instance to host applications, such as Web applications, Enterprise applications, and Web services, and resources, such as Java Message Service (JMS), and JDBC, Diagnostics.

There are two types of WebLogic Server instances: Administration Server and Managed Server.

The Administration Server instance is the central configuration controller for the entire domain. Its purpose is to:

• Host the Administration Console.

• Enable you to start and stop the servers from a central location.

• Enable you to migrate servers and services within the domain.

• Enable you to deploy applications within the domain.
There is only one Administration Server instance of WebLogic Server in a domain, and an Administration Server instance of WebLogic Server controls only one domain.

A Managed Server WebLogic Server instance is a running instance that hosts the applications and the resources that are needed by those applications. Each Managed Server WebLogic Server instance is independent of all other Managed Server WebLogic Server instances in the domain, unless they are in a cluster. You can have as many Managed Server WebLogic Server instances as you need in a domain.

The Administration Server instance of WebLogic Server stores the master copy of the domain configuration, including the configuration for all Managed Server WebLogic Server instances in the domain. Each Managed Server WebLogic Server instance stores a local copy of its configuration. When a Managed Server WebLogic Server instance starts, it connects to the Administration Server instance of WebLogic Server to synchronize the configuration.

In most cases, a single server environment is used for development purposes. This is where a single server acts as the Administration Server instance of WebLogic Server and as the host for applications, as illustrated in Figure 55-3.

However, there are some teams that use a Managed Server for either Oracle Enterprise Scheduler (ESS) runtime or Service-Oriented Architecture (SOA). When you are setting up your standalone WebLogic server, you can choose one of the following options:

- Administration Server - Oracle ADF (includes ESS libraries)
- Administration Server - Oracle ADF + ESS Runtime
- Administration Server - Oracle ADF and Managed Server - ESS Runtime
- Administration Server - Oracle ADF and Managed Server - SOA
- Administration Server and Managed Server - SOA
You deploy applications to the Administration Server instance of WebLogic Server. There are two configurations for Administration Servers:

- Non-SOA
- SOA

Not all components are available in both. For example, WebCenter libraries are not available in SOA and SOA libraries are not available in non-SOA. Oracle ADF applications containing UI must be deployed to the non-SOA WebLogic server, and SOA composites must be deployed to the SOA-configured WebLogic Server.
Some services have a Service Data Object (SDO) co-location requirement and need to be deployed to the SOA container. To deploy the service to the SOA-configured WebLogic Server, create a new EAR profile containing only that service or services from your application workspace.

For more information about SDO and SOA, see the *Developing SOA Applications with Oracle SOA Suite* guide.

For information about how to configure the SOA WebLogic Server, see *How to Set Up the Personal Environment for Standalone WebLogic Server*.

### 55.2 Running Applications on Integrated WebLogic Server

Integrated WebLogic Server is a single server that is included within JDeveloper. This server automates deploying your applications. Integrated WebLogic Server is sufficient to run your application to make sure it displays correctly in browsers, or for testing and debugging portions of the application.

**Note:**

You cannot use Integrated WebLogic Server to run SOA applications. You must deploy SOA applications to a standalone WebLogic Server instance. For more information, see *Deploying Your SOA Projects to an Administration Server Instance of WebLogic Server*.

Integrated WebLogic Server has already been configured with the Oracle Fusion Middleware Extensions for Applications domain extension templates. When the scripts are used to create the Integrated WebLogic Server and Standalone Weblogic Server domains, the Applications Core templates are configured. These templates are required for deploying and running applications using Applications Core functionality. So all of the Oracle Fusion applications will run on instances of both servers.

JDeveloper has a default connection to Integrated WebLogic Server and does not require you to manually create any deployment profiles or descriptors. Integrated WebLogic Server automates creating the same EAR that is used by Standalone WebLogic Server. You can see this when running the application. The run log shows the steps done by JDeveloper, including creating an EAR and deploying it.

When you run your application in JDeveloper using the `run` or `debug` commands, Integrated WebLogic Server automatically starts the server, packages the application to an EAR file, deploys the EAR, opens the browser, and accesses the URL for the deployed application.

When you use JDeveloper to run an application for the first time, it automatically creates the Integrated WebLogic Server instance. Make sure you have configured the domain, following the steps in *How to Use the Oracle Fusion Domain Wizard*.

You can also start the server directly from within JDeveloper. To do this, go to the main menu and select **Run > Start Server Instance**.
Tip:
The first time Integrated WebLogic Server starts, it tries to use the first available port in the 7101 - 7105 range. The following message appears as the first line in the Default server log in JDeveloper. You should use the alternate port for all access:

HTTP port conflict detected. The HTTP port will be reassigned to port 7102.

The server and the application are considered separate entities, so even if you stop the application, it does not stop the server. To terminate the application, select the application name from the terminate button dropdown menu in the Server Instance Log page, as shown in Figure 55-5.

**Figure 55-5   Terminating the Application**

To terminate the server, select the server name, as shown in Figure 55-6.

**Figure 55-6   Terminating the Integrated WebLogic Server Instance**

### 55.2.1 How to Deploy an Application with Metadata to Integrated WebLogic Server

When an application is running in Integrated WebLogic Server, the Metadata Archive (MAR) profile itself will not be deployed to a repository. Instead, a simulated Oracle Metadata Services (MDS) repository will be configured for the application that reflects the metadata information contained in the MAR. This metadata information is simulated and the application runs based on their location in source control.

Any customizations or documents created by the application are written to this simulated MDS repository directory. You can keep the default location for this directory or you can set it to a different directory. You also have the option to preserve customizations across different application runs or to delete the customizations before each application run.

Before you begin, you must first create your MAR deployment profile. For information about how to create a MAR deployment profile, see How to Create Deployment Profiles for Standalone WebLogic Server Deployment.

To deploy a MAR deployment profile to Integrated WebLogic Server:

1. Go to the Application Navigator, right-click the application and select Application Properties.

2. In the Application Properties dialog, expand Run and choose MDS. See Figure 55-7.
3. In the Run MDS page:
   a. Select the MAR profile from the MAR Profile dropdown list.
   b. Enter a directory path in Override Location if you want to customize the location of the simulated repository.
   c. Select the Directory Content option. You can choose to preserve the customizations across application runs or delete customizations before each run.

55.3 Prepare to Deploy Oracle ADF Applications to an Instance of Standalone WebLogic Server

You must prepare the application and the WebLogic Server instance before you deploy applications to an instance of Standalone WebLogic Server.

Before you begin:

Before you deploy the application to a Standalone WebLogic Server instance, you need to:

1. Create and configure the WebLogic Server domains using the Configure Fusion Domain wizard as described in Setting Up Your Development Environment.

2. If the application is using ADF Security, you may need to:
b. Migrate application-level security information to the WebLogic Server instance. For more information, see the "Configuring Security for WebLogic Server" section in the Developing Fusion Web Applications with Oracle Application Development Framework.

c. Set up JDBC URL for WebLogic Server. For more information, see the "Applications with JDBC URL for WebLogic" section in the Developing Fusion Web Applications with Oracle Application Development Framework.

d. Set up JDBC datasource for WebLogic Server. For more information, see the "Applications with JDBC Data Source for WebLogic" section in the Developing Fusion Web Applications with Oracle Application Development Framework.

e. Understand LDAP-based stores. For more information, see What You May Need to Know About Testing.

55.3.1 How to Reference the Shared Libraries

Shared libraries are available in the integrated and standalone WebLogic Server container and your projects must be updated so that they can use a shared library.

When you create your WebLogic Server domain, all the required shared libraries should be automatically created for you. When you choose a new technology or library in JDeveloper, the weblogic.xml and weblogic-application.xml files are automatically updated to reference these shared libraries. If, for some reason, the required references are not created automatically, you must update the weblogic.xml and weblogic-application.xml files manually.

The process below shows how to reference a sample oracle.shared.library shared library in a project.

To reference a shared library into a project:

1. Go to Application Navigator, expand Application Resources > Descriptors > META-INF.

2. Add reference shared library to weblogic-application.xml:
   a. Click the weblogic-application.xml file.
   b. Create a copy of the library-ref element.
      You can either accomplish this manually in the editor, or you can right-click the existing library-ref element to use the copy and paste options.
   c. Change the library-name element to oracle.shared.library. Leave the specification and implementation version values blank.

   Caution:
   Make sure that there are no blank spaces between the tag <library-ref> and the actual entry as they will cause problems.

   d. Repeat steps 2b and 2c to reference additional libraries.

3. Add reference shared library to weblogic.xml using the same steps.
55.3.2 How to Create Deployment Profiles for Standalone WebLogic Server Deployment

The deployment profiles determine how the application is bundled and deployed to Standalone WebLogic Server. When running an application within JDeveloper using Integrated WebLogic Server, these deployment profiles are not used.

Tip:
When you run your application in JDeveloper Integrated WebLogic Server, these deployment profiles are not used. Instead, JDeveloper scans the entire workspace or the current working set, (if the Run Working Set option is enabled), to construct the class loader classpaths. If the data model project is eligible to be an EJB then the Libraries and Classpath entries from that project contribute to the application root class loader. The user interface project contributes to the web application class loader.

To deploy the application, you must create deployment profiles applicable to the project or projects. The deployment profiles you need depend on your application requirements. For example, an application may include Business Components Service Interface, Web Application Archive (WAR), and MAR profiles. When you have defined these, create an EAR deployment profile for the application.

You can only deploy the application as an EAR file at the application level. Creating EAR files from the project level are incomplete and this option is disabled. The project level deployment profiles should be included in the EAR deployment profile.

Depending on the type of projects in your application, you may need to create the following deployment profiles:

- Business Components Service deployment profile. For instructions, see the "How to Deploy Web Services to Oracle WebLogic Server” section of the Developing Fusion Web Applications with Oracle Application Development Framework.

- WAR deployment profile. For instructions, see the "Creating a WAR Deployment Profile" section of the Developing Fusion Web Applications with Oracle Application Development Framework

- MAR deployment profile. For instructions, see the "Creating a MAR Deployment Profile" section of the Developing Fusion Web Applications with Oracle Application Development Framework

- EAR deployment profile. For instructions, see the "Creating an Application-Level EAR Deployment Profile" section of the Developing Fusion Web Applications with Oracle Application Development Framework

55.4 Deploying Your Oracle ADF Applications to an Administration Server Instance of WebLogic Server

You can deploy your Oracle ADF applications to a standalone WebLogic Server instance using JDeveloper.

Any necessary MDS repositories must be registered with the WebLogic Server instance. If the MDS repository is a database, the repository maps to a WebLogic Server system data source with MDS-specific requirements. Before you deploy the application, make sure to target this data source to the Administration Server instance.
of WebLogic Server. For more information about registering MDS, see the Oracle Fusion Middleware Administering Oracle Fusion Middleware guide.

---

**Note:**

If you are using the WebLogic Server Administrative Console or WLST scripts to deploy an application packaged as an EAR file that requires MDS repository configuration in `adf-config.xml`, you must run the `getMDSArchiveConfig` WLST command that returns a handle to the `MDSArchiveConfig` object for the specified archive. The returned `MDSArchiveConfig` object's `setAppMetadataRepository` and `setAppSharedMetadataRepository` methods can be used to change application and shared repository configuration in an archive. MDS configuration is required if the EAR file contains a MAR file or if the application is enabled for Design Time at Runtime. For more information about WLST commands, see the Oracle Fusion Middleware WLST Command Reference for WebLogic Server guide.

---

### 55.4.1 How to Create an Application Server Connection Using JDeveloper

To deploy your application using JDeveloper, you create a connection to the application server and then deploy the application.

**To create an application server connection:**

2. Open your application in JDeveloper.
3. Launch the Application Server Connection wizard.

   You can:

   - In the Application Server Navigator, right-click **Application Servers** and choose **New Application Server Connection**.
   - In the New Gallery, expand **General**, select **Connections** and then **Application Server Connection**, and click **OK**.
   - In the Resource Palette, choose **New > New Connections > Application Server**.

4. Complete the wizard:

   a. Enter the following information as you progress through the wizard:
   
   b. **Connection Name**: Enter a name for the connection.
   
   c. **Username** and **Password**: Enter a user name and password for the administrative user authorized to access the application server.
   
   d. **Weblogic Hostname Administration Server**: Enter the name of the WebLogic Server instance containing the TCP/IP DNS where your application (`.jar`, `.war`, `.ear`) will be deployed.
   
   e. **Port**: Enter a port number for the Oracle WebLogic Server instance on which your application (`.jar`, `.war`, `.ear`) will be deployed.
f. In the **SSL Port** field, enter an SSL port number for the Oracle WebLogic Server instance on which your application (.jar, .war, .ear) will be deployed.

g. Select **Always Use SSL** to connect to the Oracle WebLogic Server instance using the SSL port.

h. **WebLogic Domain**: Optionally enter a domain only if Oracle WebLogic Server is configured to distinguish nonadministrative server nodes by name.

i. Test the connection.

j. Click **Finish** to close the wizard and create your application server connection.

### 55.4.2 How to Deploy the Application Using JDeveloper

After you have created an application server connection and an EAR deployment profile, you can deploy the application to a standalone application server.

**To deploy an application:**

1. Deploy your project to the application server:
   
a. In the Application Navigator, right-click your application and choose **Deploy > deployment profile**.
   
b. In the Deploy wizard Deployment Action page, select **Deploy to Application Server** and click **Next**.
   
c. In the Select Server page, select the application server connection, and click **Next**.
   
d. The WebLogic Options page appears. Select a deploy option and click **Next**.

   If the adf-config.xml file in the EAR file requires MDS repository configuration, the Deployment Configuration dialog appears for you to choose the target metadata repository or shared metadata repositories, as shown in **Figure 55-8**.
The **Repository Name** dropdown list allows you to choose a target metadata repository from a list of metadata repositories registered with the Administration Server instance of WebLogic Server. The **Partition Name** dropdown list allows you to choose the metadata repository partition to which the application's metadata will be imported during deployment. For more information about managing the MDS repository, see the *Oracle Fusion Middleware Administering Oracle Fusion Middleware* guide.

**Note:**

If you are deploying an Oracle ADF application, do not use the **Deploy to all instances in the domain** option.

de. Click **Finish**.

2. Verify the run-time application as:

   http://httpHost:httpPORT/<CONTEXT>/faces/<landing jspex>

   For example, http://server06.us.company.com:7001/UIPatternsDemo/faces/ServiceRequest

55.4.3 How to Create an EAR File for Deployment

You can also use the deployment profile to create an archive file (EAR file). You can then deploy the archive file using Enterprise Manager, WLST, or the Oracle WebLogic Server Administration Console.

Although an application is encapsulated in an EAR file (which usually includes WAR, MAR, and JAR components), it may have parts that are not deployed with the EAR. For instance, ADF Business Services can be deployed as a JAR.

To create an EAR archive file:

- In the Application Navigator, right-click the application containing the deployment profile, and choose Deploy > deployment profile > to EAR file.

See How to Create Deployment Profiles for Standalone WebLogic Server Deployment.

55.5 Deploy Your SOA Projects to an Administration Server Instance of WebLogic Server

You can deploy your SOA projects using either JDeveloper or the other administration tools described in Table 55-1.

55.5.1 How to Deploy Your SOA Projects Using JDeveloper

This section discusses how to deploy your SOA projects into the Administration Server instance of WebLogic Server using JDeveloper.

The basic steps to deploying your SOA project from within JDeveloper are:

1. Define a connection. For instructions to create an application server connection, see "Creating an Application Server Connection" in the Developing SOA Applications with Oracle SOA Suite.

2. Deploy the Project. For instructions to deploy a SOA project, see "Deploying the Profile" in the Developing SOA Applications with Oracle SOA Suite.

3. Check the deployed SOA Project.

For details about deploying a SOA project, see "Deploying SOA Composite Applications" in the Developing SOA Applications with Oracle SOA Suite.

55.5.1.1 Check the Deployed SOA Project

You can check and run your deployed SOA project from two locations:

From Enterprise Manager on port 7001 by opening the Weblogic Hostname (Administration Server) URL for which you created the Connection, such as:

http://xyzzy-on.example.com:7001/em

After you log in, a screen similar to Figure 55-9 displays (the SOA tree in the left pane has been expanded and the first deployment has been selected):
Figure 55-9  Checking SOA Deployment from the EM
This chapter provides information about Oracle WebCenter Content Server (Content Server), Oracle Data Integrator (ODI), and Oracle Business Activity Monitoring (Oracle BAM) Server repositories, which are used in Oracle Fusion applications, and describes how to connect to each of these repositories using Oracle JDeveloper.

This chapter includes these sections:

- Creating a Content Repository Connection
- Creating an Repository Connection
- Creating Server Repository Connection

### 56.1 Creating a Content Repository Connection

Oracle WebCenter Content Server (Content Server), which serves as the base for the WebCenter Content system, provides a web-based repository that manages all phases of the content lifecycle from creation and approval to publishing, searching, expiration, and archiving or disposition. The Attachment component enables you to add attachments to the user interface (UI) pages that you create for Fusion web applications. Before you can implement attachments at design time in Oracle JDeveloper, you must set up a content server-based content repository connection.

For more information about Attachment components, see Implementing Attachments.

For more information about content integration, see the "Introduction to Integrating and Publishing Content" chapter of the Developing WebCenter Portal Assets and Custom Components with Oracle JDeveloper.

### 56.1.1 How to Create a Content Repository Connection

How you create a content repository connection depends upon whether your connection is for Oracle Fusion Applications development or for ad hoc development.

#### 56.1.1.1 Create a Connection for Oracle Fusion Applications Development

To set up a content server-based repository connection from an Oracle Fusion application, you run Oracle WebLogic Scripting Tool (WLST) commands to synchronize the Oracle WebLogic Server (WebLogic Server) credential store with the Content Server credential store, and then you use the Create Content Repository Connection wizard to set up a content server-based content repository connection.

**Before you begin:**

- Verify that Content Server has been deployed and you have a working server that works with the Oracle Fusion applications.
• Ensure that the code grant entry for the Attachments-Model.jar file exists in the application's jazn-data.xml file, as described in About Creating Attachments. When the application is deployed, the policies in jazn-data.xml are merged into the system-jazn-data.xml file in weblogic_server_domain_home/config/fmwconfig.

• Log in to the content server and verify that your user name is a member of the AttachmentsUser role. Note that employees and contingent workers have this role automatically.

To create a connection for Oracle Fusion Applications development:

1. Make a backup of the WebLogic Server domain's default keystore located at weblogic_server_domain_home/config/fmwconfig/default-keystore.jks.

2. Replace the weblogic_server_domain_home/config/fmwconfig/default-keystore.jks file with a copy of the domain default keystore for Content Server, which is located at content_server_domain_home/config/fmwconfig/default-keystore.jks.

3. At the command line, type the following line to start the WLST tool, if it is not currently running.

   sh jdev_install/oracle_common/common/bin/wlst.sh

   On Windows, use wlst.cmd.

4. If you have not yet connected to the server, type the following WLST command to connect to WebLogic Server, replacing the user name and password arguments with your WebLogic Server user name and password.

   connect('wls_username', 'wls_password', 'wls_uri')

   The values must be wrapped in single-quotes. The wls_uri value is typically T3://localhost:7101.

5. From WLST, execute the following commands to store the credentials. The user names and passwords must be the same as for the Content Server domain.

   When executing the commands, replace user name and password for user with the user names and passwords that are used in the WebCenter Content credentials.

   updateCred(map="oracle.wsm.security", key="keystore-csf-key", user="user name", password="password for user", desc="Keystore key")
   updateCred(map="oracle.wsm.security", key="enc-csf-key", user="user name", password="password for user", desc="Encryption key")
   updateCred(map="oracle.wsm.security", key="sign-csf-key", user="user name", password="password for user", desc="Signing key")
   exit()
Note:

If the keys do not exist, use the following commands instead:

```java
createCred(map="oracle.wsm.security", key="keystore-csf-key", user="user name", password="password for user", desc="Keystore key")
createCred(map="oracle.wsm.security", key="enc-csf-key", user="user name", password="password for user", desc="Encryption key")
createCred(map="oracle.wsm.security", key="sign-csf-key", user="user name", password="password for user", desc="Signing key")
exit()
```

6. From the Application Resources panel in JDeveloper, right-click Connections and choose New Connection > Content Repository from the menu.

7. In the Create Content Repository Connection wizard shown in Figure 56-1, complete the following information:

Create Connection In: Select Application Resources.

Connection Name: Enter FusionAppsContentRepository. Do not enter any other value. This name is copied to adf-config.xml as the primary connection for this application. If you use a different value, the name might be overwritten when installed on customer sites and cause connection problems.

Repository Type: Select Oracle Content Server.

Set as Primary Connection for Document Library: Select this checkbox.

Configuration Parameters: Enter values for the parameters listed in Table 56-1. If a parameter is not listed in the table, leave the value blank.

Contact your system administrator to obtain the correct information.

### Table 56-1  Example Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIDC Socket Type</td>
<td>jaxws</td>
</tr>
<tr>
<td>Admin Username</td>
<td>The name of a user who has been granted Content Server administration privileges.</td>
</tr>
<tr>
<td>URL of the Web Server Plugin</td>
<td>The idcnativews web service that is defined on the server. This is typically <a href="http://host:port/idcnativews">http://host:port/idcnativews</a>. Check with your system administrator.</td>
</tr>
</tbody>
</table>

Authentication Method: Select Identity propagation.
8. Click **Test Connection** and verify that the status returned is **Success**!

   **Note:**
   
   If the test is unsuccessful, verify that the values that you entered are correct and try again.

9. Click **OK**.

10. From the Application Resources panel, expand **Connections** to see the new content repository connection, as shown in **Figure 56-2**.

11. Complete the following steps to verify that the application contains all required libraries and jazn-data.xml entries.

   a. Create a new JavaServer Faces page named **DocumentLibrary.jspx**.
   
   b. From the Applications Resources panel, select the content repository, drag it onto the JavaServer Faces page, and choose **Document Manager**.
   
   c. Delete **DocumentLibrary.jspx**.
56.1.1.2 Create a Connection for Ad Hoc Development

Sometimes you might need a quick connection for prototyping or assessment purposes and you do not want to use the central WebCenter Content environment. In this situation, use the Create Content Repository Connection wizard to create a connection in your own environment.

Before you begin:

- Ensure that the code grant entry for the Attachments-Model.jar file exists in the application's jazn-data.xml file, as described in About Creating Attachments. When the application is deployed, the policies in jazn-data.xml are merged into the system-jazn-data.xml file in weblogic_server_domain_home/config/fmwconfig.

- Ensure that the sockets on the WebCenter Content Managed Server are enabled.

- You need the following information to create a content repository connection:
  - Host name of the machine that is running the WebCenter Content Managed Server.
  - WebCenter Content Managed Server listener port.
  - URL for the login page to the content server instance.
  - WebCenter Content shared credentials user name and password

  Contact your system administrator to obtain this information.

To set up a content repository connection for ad hoc development:

1. From the Application Resources panel in JDeveloper, right-click Connections and choose New Connection > Content Repository from the menu.

The Create Content Repository Connection wizard is displayed, as shown in Figure 56-3.
2. Complete the following information:

**Create Connection In:** Select *Application Resources*.

**Connection Name:** Enter `FusionAppsContentRepository`. Do not enter any other value. This name is copied to `adf-config.xml` as the primary connection for this application. If you use a different value, the name might be overwritten when installed on customer sites and cause connection problems.

**Repository Type:** Select *Oracle Content Server*.

**Set as Primary Connection for Document Library:** Select this checkbox.

**Configuration Parameters:** Enter values for the parameters shown in Table 56-2. Except for the RIDC Socket Type parameter, all values shown are examples only. If a parameter is not listed in the table, leave the value blank.

Contact your system administrator to obtain the correct information.

### Table 56-2 Example Configuration Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIDC Socket Type</td>
<td>socket</td>
</tr>
<tr>
<td>Server Hostname</td>
<td>abc.example.com</td>
</tr>
<tr>
<td>Content Server Listener Port</td>
<td>4444</td>
</tr>
</tbody>
</table>

**Authentication:** Select *External Application*. Click the *Add* icon and complete the following information to create a new External Application:
a. Enter a unique name for the **External Application**. Click **Next** to continue to Step 2.

b. Enter the following information, as shown in **Figure 56-4**.
   - **Login URL**: Paste the URL for the login page. (Contact your system administrator for this information). For example:
     - `http://abc.example.com:2244/abc/abcplg?
       AbcService=LOGIN&Action=GetTemplatePage&Page=HOME_PAGE&Auth=Internet`
   - **User Name/ID Field Name**: Enter the field name for the user name or ID, such as `username`.
   - **Password Field Name**: Enter the field name for the password, such as `password`.
   - **Tip**: The User Name and Password field names are derived from the HTML input field names.

![Figure 56-4 Register External Application Wizard — General (Step 2)](image)

c. Click **Next** to continue to Step 3 and click **Next** to continue to Step 4.

d. Complete the following information:
   - **Specify Shared Credentials**: Select this checkbox.
   - **User Name** and **Password**: Enter the Shared Credentials user name and password. (Contact your system administrator for this information).
   - Click **Next** to continue to Step 5.

e. Complete the following information:
   - **Specify Public Credentials**: Select this checkbox.
   - **User Name** and **Password**: Enter the Shared Credentials user name and password. (Contact your system administrator for this information).
Click Finish to save your entries, create the External Application, close the wizard, and return to the Create Content Repository Connection page.

**Authentication:** Choose the newly created External Application from the dropdown list.

3. Click **Test Connection** and verify that the status returned is **Success**.

**Note:**
If the test is unsuccessful, verify that the values that you entered are correct and try again.

4. Click **OK**.

5. From the Application Resources panel, expand **Connections** to see the new content repository connection, as shown in Figure 56-5.

![Figure 56-5](image)

**Figure 56-5**  **JDeveloper — Application Navigator**

6. Complete the following steps to verify that the application contains all required libraries and jazn-data.xml entries.


   b. From the Applications Resources panel, select the content repository, drag it onto the JavaServer Faces page, and choose **Document Manager**.

   c. Delete `DocumentLibrary.jspx`.

**56.1.2 Troubleshooting Content Server Connections**

Exceptions can occur when the connection is improperly configured. The four most common exceptions are the following errors:

- Insufficient user privileges error
- WS-Security header processing error
- Access denied due to incorrect access grants for the application
- Access denied error due to incorrect access grants for attachments

For more information, see the "Diagnosing Problems" chapter in the *Administering Web Services*.

**56.1.2.1 User Does Not have Sufficient Privileges**

If you are sure that the user is a member of the AttachmentsUser role, then you must consult logs for the cause of the insufficient privileges error message. The most common cause is a blank or invalid signature, but this exception can be the consequence of several different misconfiguration issues.
To diagnose the problem, enable applications logging for the
oracle.apps.fnd.applcore.attachments.model.% module with a logging
level of FINEST. Search the application log files for the string "Unable to generate
digital signature." The stack trace might indicate the cause. For example, if it reports
that the "keystore has been tampered with, or the password is wrong," verify that the
password in credentials store for the application's server domain matches the
password in the credentials for the content server's domain. If the stack trace reports
"Access Denied," verify that the code grants described in About Creating Attachments
have been added to the jazn-data.xml file. For information about applications
logging, see the "Troubleshooting Oracle Fusion Applications Using Incidents, Logs,
QuickTrace, and Diagnostic Tests" chapter in the Administrator's Guide.

If your search through the application logs does not find an "Unable to generate digital
signature" string, the cause might be that the content server cannot verify the digital
signature. To diagnose the problem, set up tracing for fusionappsattachments. Be
sure to enable full verbose tracking and enable save. Access the system audit
information server output and search for XFND_SIGNATURE. A blank signature
indicates that the signature was not generated by the Oracle Fusion application. If the
signature is not blank and the "$DefaultCheckinSigningScheme: Signature Verification
Failed" message exists, the cause might be that the credentials for the application's
server domain do not match the credentials for the content server's domain and you
need to repeat the connection steps described in this section for the appropriate
application type.

Related Links
The following documents provide additional information related to the topics
discussed in this section:

- For more information about setting up tracing for fusionappsattachments,
  see the "System Audit Tracing Sections Information" section in the Oracle

- For more information about system audit, see the "System Audit Information"
  section in the Administering Oracle WebCenter Content.

56.1.2.2 Invalid Security: Error in Processing the WS-Security Header
The common cause for the WS-Security header processing exception is that global
policy attachment (GPA) has not been set up.

56.1.2.3 Access Denied: Policy Enforcement Exception
If you receive the following message, then your application does not have the correct
access grants.

oracle.fabric.common.PolicyEnforcementException: access denied
(oracle.wsm.security.WSIdentityPermission resource=AppName assert)

If you are using the file-based policy store for grants, then ensure that you have a
<grant> element similar to Example 56-1 in the <jazn-policy> element in the
jazn-data.xml file. Replace application-short-name with the name for your
application. Note that the application short name appears as the resource value in
the error message.

Example 56-1 Application Permissions Entry

<grant>
  <grantee>
    <codesource>
56.1.2.4 Access Denied: Credential AccessPermission

The following exception typically indicates that the code grants described in About Creating Attachments have not been added to the jazn-data.xml file or have not been merged into the system-jazn-data.xml file in weblogic_server_domain_home/config/fmwconfig.

access denied (oracle.security.jspservice.credstore.CredentialAccessPermission context=SYSTEM, mapName=oracle.wsm.security, keyName=enc-csf-key read)

56.2 Creating an Oracle Data Integrator Repository Connection

Oracle Data Integrator (ODI) combines all the elements of data integration—data movement, data synchronization, data quality, data management, and data services—to ensure that information is timely, accurate, and consistent across complex systems.

ODI is built on several components all working together around a centralized metadata repository. The ODI architecture is organized around a modular repository, which is accessed in client-server mode by components.

The Oracle Fusion Applications ODI repository consists of a master repository and a work repository. The master repository contains the security information, the topology information (definitions of technologies and servers), and the versions of the objects. A work repository stores information for:

- **Models** — Includes datastores, columns, data integrity constraints, cross-references, and data lineage.

- **Projects** — Includes declarative rules, packages, procedures, folders, knowledge modules, and variables.

- **Runtime information** — Includes scenarios, scheduling information, and logs.

You use Oracle Data Integrator Studio to access the repositories; administer the infrastructure; reverse-engineer the metadata; develop projects; and perform scheduling, operating, and monitoring executions.

For information about the role of ODI in securing Oracle Fusion applications, see c__2798.

**Related Links**

The following document provides more information about the topics discussed in this section:

- For more information about connecting to the ODI master and work repositories, see the "Administering Repositories" chapter in the Oracle Fusion Middleware Developing Integration Projects with Oracle Data Integrator.
56.3 Creating Oracle Business Activity Monitoring Server Repository Connection

Oracle Business Activity Monitoring (Oracle BAM) provides an active data architecture that dynamically moves real-time data to end users through every step of the process. This solution actively collects data, applies rules designed to monitor changes, and delivers the information in reports to users.

You must create a connection to Oracle BAM Server to browse the available data objects and construct transformations while you are designing your applications. When the application is running, the Oracle BAM Server connection is used to publish data to Oracle BAM data objects. Only one Oracle BAM Server connection per Business Process Execution Language (BPEL) project is currently supported. For more information about BPEL integration, see the Developing SOA Applications with Oracle SOA Suite guide.

When building an application in JDeveloper, the methods of connecting to Oracle BAM Server are:

- Oracle BAM Adapter in a service-oriented architecture (SOA) composite application
- Oracle BAM sensor actions in a BPEL process

56.3.1 How to Create an Oracle BAM Connection

Use the BAM Connection wizard to create a connection to Oracle BAM Server.

**Note:**

Do not create an Oracle BAM Server connection through the Resource Palette that displays when you select View > Resource Palette. Create Oracle BAM Server connections from the Application Resources panel, either directly or by copying an existing connection from the Resource Catalog.

**To create an Oracle BAM Server connection:**

1. In JDeveloper, select New from the File menu to open the New Gallery dialog.

2. Select the General > Connections category, then select BAM Connection. Click OK to open the Oracle BAM Connection wizard, as shown in Figure 56-6.
3. Enter a unique name to identify this connection.

4. Select Application Resources and click Next.

5. Enter the following connection information:

   **BAM Web Host**: Enter the name of the host on which Oracle BAM Report Server and the web server are installed. In most cases, the web host and the server host are the same.

   **BAM Server Host**: Enter the name of the host on which the Oracle BAM Server is installed.

   **User Name** and **Password**: Enter the Oracle BAM Server user name and password. The user name is typically `bamadmin`.

   **HTTP Port**: Enter the port number or accept the default value of 9001. This is the HTTP port for the Oracle BAM Web Host.

   **JNDI Port**: Enter the port number or accept the default value of 9001. The Java Naming and Directory Interface (JNDI) port is for Oracle BAM Report Cache, which is part of Oracle BAM Server.

   **Use HTTPS**: Select this option if you want to use HTTP with Secure Sockets Layer (HTTPS) to connect to the Oracle BAM Server during design time.

6. Click Next.

7. Click Test Connection to ensure that the connection is properly configured.

8. Click Finish.

### 56.3.2 How to Use Oracle BAM Adapter in a SOA Composite Application

Oracle BAM Adapter is a Java EE Connector Architecture (JCA) compliant adapter, which can be used from a Java EE client to send data and events to Oracle BAM Server. Oracle BAM Adapter supports the following operations on Oracle BAM data...
objects: inserts, updates, upserts, and deletes. Oracle BAM Adapter can perform these operations over Enterprise JavaBeans (EJB) calls or over Simple Object Access Protocol (SOAP), all configurable in JDeveloper.

The Oracle BAM Adapter supports batching of operations, but behavior with batching is different from behavior without batching. In general, the Oracle BAM sensor action is not part of the BPEL transaction. When batching is enabled, BPEL does not wait for an Oracle BAM operation to complete. It is an asynchronous call.

When batching is disabled, BPEL waits for the Oracle BAM operation to complete before proceeding with the BPEL process, but it will not roll back or stop when there is an exception from Oracle BAM. The Oracle BAM sensor action logs messages to the same sensor action logger as BPEL.

Oracle BAM Adapter provides three mechanisms by which you can send data to Oracle BAM Server in your SOA composite application as you develop it in JDeveloper.

- Oracle BAM Adapter can be used as a reference binding component in a SOA composite application. For example, Oracle Mediator can send data to Oracle BAM using Oracle BAM Adapter.
- Oracle BAM Adapter can also be used as a partner link in a BPEL process to send data to Oracle BAM as a step in the process.
- Oracle BAM sensor actions can be included within a BPEL process to publish event-based data to Oracle BAM data objects.

For more information about using Oracle BAM Adapter in a SOA composite application, see the "Integrating Oracle BAM with SOA Composite Applications" chapter in the Developing SOA Applications with Oracle SOA Suite.

### 56.3.3 How to Integrate Sensors With Oracle BAM

You can create sensor actions in Oracle BPEL Process Manager to publish sensor data as data objects on Oracle BAM Server. When you create the sensor action, you can select an Oracle BPEL Process Manager variable sensor or activity sensor that you want to get the data from, and also the data object in Oracle BAM Server in which you want to publish the sensor data.

For more information about integrating sensors with Oracle BAM, see the "Integrating Oracle BAM with SOA Composite Applications" chapter in the Developing SOA Applications with Oracle SOA Suite.
This chapter describes how to define a profile, which is a set of changeable options that affect the way your application looks and behaves. Profiles control how Oracle Fusion Applications operate for users by the values that are set. Profiles can be set at different levels depending on how the profiles are defined.

This chapter includes the following sections:

- Introduction to Profiles
- About Integrating Profiles Task Flows into
- About Setting and Accessing Profile Values
- About Managing Profile Definitions
- About Managing Profile Categories

### 57.1 Introduction to Profiles

Profiles are permanent user preferences and system configuration options. They allow for the centralized management of configuration data but with sophisticated, customized user, security, and session-context access to the values. The Profile Service application programming interface (API) provides the access to profile values.

Hierarchies enable system administrators to group and set profiles according to their business needs. The hierarchy is fixed in the profile definition, you cannot just randomly mix and match the levels. For more information about profile levels, see About Managing Profile Definitions.

Users may be able to set their own profile options, depending on settings in the profile definition. However, not all profiles are visible to end-users, and some profiles, while visible, may not be updated by end-users.

Categories can be used to group profiles based on their functional area. Administrators can categorize profiles and then easily search on profiles by category.

**When to define a profile:**

- Evaluate if there is a genuine need for an option before creating a profile. Do not force the customer to make a decision about an issue that is of no concern to them.
- Look for duplicate or similar profiles, even in other products, before creating a new one. For example, you do not need multiple profiles to choose a preferred currency.
- Do not use profiles to cache temporary session attributes. Profiles are permanent user preferences and system configuration options.
Every Oracle application registers task flows with a product called Oracle Fusion Functional Setup Manager. Functional Setup Manager provides a single, unified user interface that allows customers and implementers to configure all Oracle applications by defining custom configuration templates or tasks based on their business needs.

The Functional Setup Manager user interface (UI) enables customers and implementers to select the business processes or products that they want to implement.

Function Security controls your privileges to a specific task flow, and users who do not have the required privilege cannot view the task flow. For more information about how to implement function security privileges and roles, see Implementing Function Security.

For information about implementing your specific product family, do the following:

1. Access the Oracle Fusion Applications library.

2. See the Implementing Common Features guides for your product family.

Table 57-1 lists the task flows related to profiles and their parameters.

Table 57-1 Profiles Task Flows and Parameters

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Parameters Passed</th>
<th>Behavior</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Administrator Profile</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/profiles/ui/flow/ManageAdminProfileValuesTF.xml#profileValues_task-flow-definition</td>
<td>mode='search' [moduleType] [moduleKey] [categoryName] [categoryApplicationId] mode='edit' profileOptionName [pageTitle]</td>
<td>Search and edit all profile values for a system administrator. To search all profiles, do not pass any parameters. To search all profiles in a module, pass moduleType/moduleKey To search all profiles in a category, pass categoryName/categoryApplicationId. moduleType/moduleKey and categoryName/categoryApplicationId are mutually exclusive and cannot be passed in together.</td>
<td>Search and edit all profile values for a system administrator. To search all profiles, do not pass any parameters. To search all profiles in a module, pass moduleType/moduleKey To search all profiles in a category, pass categoryName/categoryApplicationId. moduleType/moduleKey and categoryName/categoryApplicationId are mutually exclusive and cannot be passed in together.</td>
</tr>
</tbody>
</table>
### Table 57-1  (Cont.) Profiles Task Flows and Parameters

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Parameters Passed</th>
<th>Behavior</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Profile Categories</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/profiles/ui/flow/ManageProfileCategoriesTF.xml#profileCategories_task-flow-definition</td>
<td>mode='search' [moduleType] [moduleKey]</td>
<td>Search and edit profile categories.</td>
<td>Search and edit profile categories.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode='edit' name applicationId [pageTitle]</td>
<td>To search all profile categories, do not pass any parameters.</td>
<td>To search all profile categories, do not pass any parameters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>To search all profile categories in a module, pass in moduleType/moduleKey.</td>
<td>To search all profile categories in a module, pass in moduleType/moduleKey.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>To edit a specific profile category, pass in name/applicationId.</td>
<td>To edit a specific profile category, pass in name/applicationId.</td>
</tr>
<tr>
<td>Manage Profile Options</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/profiles/ui/flow/ManageProfileOptionsTF.xml#profileOptions_task-flow-definition</td>
<td>mode='search' [moduleType] [moduleKey]</td>
<td>Search and edit profile definitions.</td>
<td>Search and edit profile definitions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>mode='edit' profileOptionName [pageTitle]</td>
<td>In 'search' mode:</td>
<td>In 'search' mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>To search all profile options, do not pass any parameters.</td>
<td>To search all profile options, do not pass any parameters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>To search all profile options in a module, pass in moduleType/moduleKey.</td>
<td>To search all profile options in a module, pass in moduleType/moduleKey.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>In 'edit' mode:</td>
<td>In 'edit' mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>To edit a specific profile option, pass in profileOptionName.</td>
<td>To edit a specific profile option, pass in profileOptionName.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If mode is not explicitly passed, the default is 'search'.</td>
<td>If mode is not explicitly passed, the default is 'search'.</td>
</tr>
</tbody>
</table>

### 57.3 About Setting and Accessing Profile Values

You can set profile values using the Setup UI, and access them programmatically or by using expression language.

### 57.3.1 How to View and Set Profile Values Using the Setup UI

You can use the Profile Option Values page to view the profile values. The page is shown in Figure 57-1.
Note:

- Any change you make to a profile option has an immediate effect on the way your applications run for that session. And, when you log on again, changes you made to your User-level options in a previous session are still in force.
- When a profile value is changed, the user setting the value will always see the update immediately. Other users may not see the changed value until logging out and back in.

To view or edit profile values, do the following:

1. Go to the Manage Profile Option Values page to search for the required profile option.

   **Figure 57-1  Manage Profile Option Values Page**

2. Enter your search criteria, then click **Search**.

   **Note:**
   You can also select **Reset** to clear your entries and start again, or **Save** to save the entries for a future search.

3. In the **Search Results: Profile Options** section, highlight the required profile option.

4. In the **Profile Values** section, add a new value or delete an existing one.
Create a new row for every value set for this profile for every level/level value pair. The Profile Value is the value that has been defined in the profile definition’s SQL Validation.

57.3.2 About Accessing Profile Values Programmatically

The ProfileServiceAM API can be found in the following package:

oracle.apps.fnd.applcore.profiles.profileService.applicationModule

Before you can use this profile, you must add the Applications Core library to your Model project. For more information, see Adding Necessary Libraries to Your Data Model Project.

Import the Profile class and call the Profile.get() method to get the values for the profile name provided.

For example:

```java
import oracle.apps.fnd.applcore.Profile;
...
fnfDiagnostics = Profile.get("AFLOG_ENABLED");
```

57.3.3 About Accessing Profile Values Using Expression Language

Accessing a profile value using expression language (EL) simply requires defining the oracle.apps.fnd.applcore.Profile managed bean with the name Profile in the adfc-config.xml file at requestScope. See Figure 57-2.

![Edit adfc-config.xml File](image)

After the bean is defined, you can refer to any profile value as:

```java
# {Profile.values.PROFILE_OPTION_NAME}
```

57.4 About Managing Profile Definitions

You can update profile definitions using either Functional Setup, or a standalone "super-web" type UI Shell page that embeds calls to the task flow directly in the menu.

When defining profile definitions you also define profile levels, which are part of a hierarchy. When working with profile levels, carefully consider the levels you enable for your profiles. Only enable them at the levels that make sense. You do not want end-users changing settings for profiles that they do not understand. Currently, only the following hierarchy is available:

- Site (lowest level)
- Product
• User (highest level)

As most profiles are user preferences and can potentially be set at these three levels, this is the default hierarchy. Profiles can be set at one or more levels.

---

**Note:**

A higher-level profile value overrides a lower-level value.

---

**Table 57-2** describes how profile settings are used:

<table>
<thead>
<tr>
<th>Hierarchy</th>
<th>Level</th>
<th>Profile Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (Lowest)</td>
<td>Site</td>
<td>All users at an installation site.</td>
</tr>
<tr>
<td>2</td>
<td>Product</td>
<td>This level is intended to be the product owning the current code module. The product level is only available if it has been set on the session. Typically this is in a servlet filter, but it may be in other places in other technologies.</td>
</tr>
<tr>
<td>3 (Highest)</td>
<td>User</td>
<td>An individual user, identified by their UserID (UserGUID) for the current session.</td>
</tr>
</tbody>
</table>

When a profile is set at more than one level, Site has the lowest priority, superseded by Product, with User having the highest priority. A value entered at the Site level may be overridden by values entered at any other level. A value entered at the User level has the highest priority and overrides values at any other level.

For example, assume the Printer profile is set only at the Site and Product levels. When a user logs on, the Printer profile assumes the value set at the Product level, since it is the highest-level setting for the profile.

---

**Tips:**

- System administrators should set site-level profile values before specifying values at any other level.

- The profile values specified at the site-level work as defaults until profile values are specified at the other levels. Profiles are enterprise-striped. In a multi-tenant environment, VPD policies restrict the profile values to only those defined in the relevant enterprise. Consequently, in a multi-tenant environment, a site-level profile value behaves like an enterprise-level profile value.

---

### 57.4.1 How to Edit Profile Definitions

You can use the Profile editor to update profile definitions. **Figure 57-3** shows the Manage Profiles Options page.

To edit profile definitions, do the following:

1. Go to the Manage Profiles Options page to search for the profile that you want to update.
2. In the **Search Results: Profile Options** section, highlight a profile option and do any of the following:
   - Use the **Actions** or **View** options
   - **Create** a new profile option
   - **Edit** an existing profile option. The Edit window is shown in **Figure 57-4**.

![Manage Profiles Options Page](image)

![Edit Profile Option](image)

3. In the **Search Results: Profile Option Levels** section, do any of the following:
   - **Delete** an existing profile option
   - **Detach** a profile option to open it in a new window
• Create a new profile level

• Enable or disable user access to this profile:
  
  – **Enabled**: Select this option to allow user access.
  
  – **Updateable**: Select this option to give the user update privileges. Leave unselected if you want the user to have read-only access. This option is disabled unless the **Enabled** option is selected.

  Enabling the profile for end-user access allows the user to set their own values.

  **Note:**
  
The **Enabled** and **Updateable** check boxes determine whether you can read or write (respectively) values at that level.

• Delete an existing profile level

  **Note:**
  
Deleting a Profile Option level (or never creating one) is effectively the same as disabling it.

• Detach the profile levels child table to open it in a new window

### 57.4.2 About Registering a New Profile Option

When registering a new profile option for a profile definition, one of the key properties is the SQL validation property. If the values for a profile option are limited to a discrete list from which to choose, the SQL validation property must be set.

• It must be a valid SQL statement that selects two columns.

• The first column should be the display value that the administrator will see in the Manage Profile Option Values task flow. This column can be a translated value if appropriate for the particular profile option.

• The second column should be the code or ID that the product business logic will understand how to process. For example:

```
SELECT MEANING, LOOKUP_CODE
FROM FND_LOOKUPS
WHERE LOOKUP_TYPE = 'YES_NO'
```

### 57.5 About Managing Profile Categories

Grouping profiles into categories makes them easier to find because **Category** is the main driver when searching for profiles. Group profiles into categories that make sense, such as categories based on their functional areas. Categories can be used to search for related profiles in the Administration UIs and also for defining data security rules. You can use the Manage Profile Categories editor to add new categories or add profiles to an existing category.
The grouping is many to many, which means that profiles can be in more than one category and categories can have more than one profile. The basic guideline for grouping profiles is that profiles affecting the same feature, or profiles an Administrator would likely want to see at the same time, should all be in the same category. Oracle seeds a number of categories out of the box; customers are free to create their own or edit those that are shipped.

57.5.1 How to Manage Profile Categories

Like profile definitions, you can manage profile categories using either Functional Setup, or a standalone "super-web" type UI Shell page that embeds calls to the task flow directly in the menu.

To manage profile categories, do the following:

1. Go to the Manage Profile Categories page and search for the required profile category, as shown in Figure 57-5.

**Figure 57-5 Manage Profile Categories Page**

2. In the **Search Results: Profile Categories** section, highlight the required profile category and do any of the following:
   - Use the **Actions** or **View** options
   - **Create** a new profile category
   - **Edit** an existing profile category. The Edit window is shown in Figure 57-6.
Figure 57-6  Edit Profile Category

- Delete an existing profile category
- Detach a profile category to open it in a new window

3. In the Search Results: Profile Options section, do any of the following:
   - Use the Actions or View options
   - Create a new profile option
   - Delete an existing profile category
   - Detach a profile category to open it in a new window
This chapter discusses the Seed Data Loader and using it from within Oracle JDeveloper.

This chapter includes the following sections:

- Introduction to the Seed Data Loader
- Using the Seed Data Loader in JDeveloper
- Translating Seed Data

58.1 Introduction to the Seed Data Loader

Application Seed Data is the essential data to enable Oracle Fusion applications. Some examples include static lists of values, functional or error messages and lookup values. Seed data is generally static in nature, although it is possible for customers to customize some seed data values after delivery. Any non-transactional data values loaded into a database at customer delivery time can be considered seed data.

Seed data is extracted from Oracle development databases at design time into external files. These files are delivered to the customer and uploaded to the customer’s database. Seed data can be delivered and installed at any point in the application lifecycle, such as for a new installation, a major or minor release upgrade, or a patch/change delivery.

Applications that manage seed data need to have a certain amount of knowledge about the seed data. This is so that data to be recreated on the target database is loaded to the correct tables, while preserving referential integrity. This seed data knowledge, or seed meta-data, also must be delivered in some form along with the extracted seed data files. This meta-data drives how the data is extracted and uploaded.

The Seed Data Utility, which runs only under JDeveloper, will provide data extraction from the development instances of Oracle Fusion applications. It will also load the extracted data to the customer database instances of Oracle Fusion applications, by integrating with Oracle Fusion Applications Patch Manager.

Note:

Each entity type, such as Profile and Messages, will have its own dedicated utility. See Table 58-1.
58.2 Using the Seed Data Loader in JDeveloper

The Seed Data Extract and Upload processes are run directly from the Seed Data-configured Oracle Application Development Framework (Oracle ADF) Business Components application modules. The Seed Data Configuration, Extract, and Upload processes are all run from within JDeveloper, the same development environment in which the Business Object components are defined.

58.2.1 Introduction to the Seed Data Framework

The Seed Data Framework is delivered as a plug-in extension to the JDeveloper environment. The Seed Data Framework is installed by default; there are no other installation or setup steps to perform to begin using the Seed Data Framework tasks. There are support libraries that need to be present in the Business Component project class path before running the Seed Data tasks. See How to Set Up the Seed Data Environment.

Available Seed Data Loaders

The loaders and view objects listed in Table 58-1 are supported.

Legend

- **Loader**: Location of a seed loader-enabled application module. Assume a prefix of oracle.apps.fnd.applcore.
- **VO**: Driving view object.
- **Is the Module Striped?**: Does this view object require a module argument?

<table>
<thead>
<tr>
<th>Loader</th>
<th>View Object</th>
<th>Is the Module Striped?</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachments.attachmentLoader.applicationModule.attachmentsLoaderAM</td>
<td>FndDocumentEntitiesVL</td>
<td>N</td>
</tr>
<tr>
<td>attachments.attachmentLoader.applicationModule.attachmentsLoaderAM</td>
<td>FndDocumentCategoriesVL</td>
<td>Y</td>
</tr>
<tr>
<td>attachments.attachmentLoader.applicationModule.attachmentsLoaderAM</td>
<td>FndDocCategoriesToEntitiesVO</td>
<td>N</td>
</tr>
</tbody>
</table>

**Note**: Before running this loader, you need to run the first two loaders listed in this table that have the FndDocumentEntitiesVL and the FndDocumentCategoriesVL view objects.

<table>
<thead>
<tr>
<th>Loader</th>
<th>View Object</th>
<th>Is the Module Striped?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loader</td>
<td>View Object</td>
<td>Is the Module Striped?</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>dataSecurity.dataSecurityService.applicationModule.DataSecurityAM</td>
<td>FndFormFunctions</td>
<td>Y</td>
</tr>
</tbody>
</table>
| docseq.docSeqService.applicationModule.DocSeqServiceAM | There are four separate VOs for this, one for each determinant type:  
- DocumentSequencesLoader  
- DocumentSequencesBULoader  
- DocumentSequencesLedgerLoader  
- DocumentSequencesLegalLoader  
Use the loader appropriate for the determinant type of your sequence, or all four if you want to download all document sequences for all determinant types. | Y |
| docseq.docSeqService.applicationModule.DocSeqServiceAM | DocSequenceCategories | Y |
| docseq.docSeqService.applicationModule.DocSeqServiceAM | DocSequenceAudit | N |
| docseq.docSeqService.applicationModule.DocSeqServiceAM | DocSequenceUsers | N |
| flex.dff.category.categoryService.applicationModule.CategoryServiceAM | Category | N |
| flex.dff.descriptiveFlexfieldService.applicationModule.DescriptiveFlexfieldServiceAM | DescriptiveFlexfield | Y |
| flex.dff.descriptiveFlexfieldService.applicationModule.DescriptiveFlexfieldServiceAM | DescriptiveFlexfieldSecondaryUsage | Y |
| flex.kff.keyFlexfieldService.applicationModule.KeyFlexfieldServiceAM | KeyFlexfield | Y |
| flex.kff.keyFlexfieldService.applicationModule.KeyFlexfieldServiceAM | KeyFlexfieldSecondaryTableUsage | Y |
| flex.vst.valueSetService.applicationModule.ValueSetServiceAM | ValueSet | Y |
| lookups.lookupService.applicationModule.LookupServiceAM | LookupView1 | Y |
| lookups.lookupService.applicationModule.LookupServiceAM | StandardLookupType1 | Y |
### Table 58-1 (Cont.) Available Seed Data Loaders

<table>
<thead>
<tr>
<th>Loader</th>
<th>View Object</th>
<th>Is the Module Striped?</th>
</tr>
</thead>
<tbody>
<tr>
<td>lookups.lookupService.applicationModule.LookupServiceAM</td>
<td>CommonLookupType1</td>
<td>Y</td>
</tr>
<tr>
<td>lookups.lookupService.applicationModule.LookupServiceAM</td>
<td>SetIdLookupType1</td>
<td>Y</td>
</tr>
<tr>
<td>messages.messageService.applicationModule.MessageServiceAM</td>
<td>Message</td>
<td>Y</td>
</tr>
<tr>
<td>nls.currencyService.applicationModule.CurrencyServiceAM</td>
<td>Currency</td>
<td>N</td>
</tr>
<tr>
<td>nls.isoLanguageService.applicationModule.IsoLanguageServiceAM</td>
<td>IsoLanguage</td>
<td>N</td>
</tr>
<tr>
<td>nls.languageService.applicationModule.LanguageServiceAM</td>
<td>Language</td>
<td>N</td>
</tr>
<tr>
<td>nls.naturalLanguageService.applicationModule.NaturalLanguageServiceAM</td>
<td>NaturalLanguage</td>
<td>N</td>
</tr>
<tr>
<td>nls.territoryService.applicationModule.TerritoryServiceAM</td>
<td>Territory</td>
<td>N</td>
</tr>
<tr>
<td>nls.timezoneService.applicationModule.TimezoneServiceAM</td>
<td>Timezone</td>
<td>N</td>
</tr>
<tr>
<td>profiles.profileService.applicationModule.ProfileServiceAM</td>
<td>ProfileCategory</td>
<td>Y</td>
</tr>
<tr>
<td>profiles.profileService.applicationModule.ProfileServiceAM</td>
<td>ProfileHierarchy</td>
<td>N</td>
</tr>
<tr>
<td>profiles.profileService.applicationModule.ProfileServiceAM</td>
<td>ProfileLevel</td>
<td>N</td>
</tr>
<tr>
<td>profiles.profileService.applicationModule.ProfileServiceAM</td>
<td>ProfileOption</td>
<td>Y</td>
</tr>
<tr>
<td>ref.industryService.applicationModule.IndustryServiceAM</td>
<td>Industry</td>
<td>N</td>
</tr>
<tr>
<td>setid.setIdservice.applicationModule.SetIdserviceAM</td>
<td>SetIdservice</td>
<td>N</td>
</tr>
<tr>
<td>setid.setIdservice.applicationModule.SetIdserviceAM</td>
<td>SetIdSummary</td>
<td>Y</td>
</tr>
<tr>
<td>Loader</td>
<td>View Object</td>
<td>Is the Module Striped?</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------</td>
<td>------------------------</td>
</tr>
</tbody>
</table>
| setid.setIdService.applicationModule.SetIdServiceAM | There are five separate VOs for this, one for each determinant type:  
  - SetIdAssignmentsAB  
  - SetIdAssignmentsBU  
  - SetIdAssignmentsCST  
  - SetIdAssignmentsPU  
  - SetIdAssignmentsSet  
  Use the loader appropriate for the determinant type of your sequence, or all five if you want to download all document sequences for all determinant types. | N                      |
| setid.setIdService.applicationModule.SetIdServiceAM | SetIdReferenceGroup         | Y                      |
| trees.loader.applicationModule.TreeStructureLoader | FndTreeStructure            | Y                      |
| taxonomy.taxonomyService.applicationModule.App1TaxonomyAM | ApplTaxonomyVO              | Special case - all the data extracted into a single file. |                      |
| taxonomy.taxonomyService.applicationModule.App1TaxonomyAM | ApplTaxonomyHierarchyVO     | Special case - all the data extracted into a single file. |                      |
| taxonomy.taxonomyService.applicationModule.App1TaxonomyAM | ApplTaxonomyPVO             |                      |
| taxonomy.taxonomyService.applicationModule.App1TaxonomyAM | ApplTaxonomySeedDataVO      |                      |
| taxonomy.taxonomyService.applicationModule.App1TaxonomyAM | ApplTaxonomySeedDataPVO     |                      |
| taxonomy.taxonomyService.applicationModule.App1TaxonomyAM | ApplTaxonomyComponentsVO    | Y                      |
Table 58-1 (Cont.) Available Seed Data Loaders

<table>
<thead>
<tr>
<th>Loader</th>
<th>View Object</th>
<th>Is the Module Striped?</th>
</tr>
</thead>
<tbody>
<tr>
<td>taxonomy.taxonomyService.applicationModule.App</td>
<td>ApplTaxonomyNodeComponentsVO</td>
<td>Y</td>
</tr>
<tr>
<td>lTaxonomyAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>trees.loader.applicationModule.DefaultTreeLoad</td>
<td>FndTree</td>
<td>N</td>
</tr>
<tr>
<td>er</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oracle.apps.fnd.appltest.diagfwk.seeddata.mode</td>
<td>FndDiagTag</td>
<td>Y</td>
</tr>
<tr>
<td>l.DiagFwkSeedDataAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oracle.apps.fnd.appltest.diagfwk.seeddata.mode</td>
<td>FndDiagTest</td>
<td>Y</td>
</tr>
<tr>
<td>l.DiagFwkSeedDataAM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

58.2.2 How to Set Up the Seed Data Environment

To set up the Seed Data environment, follow these steps in JDeveloper before starting the Seed Data Framework tasks:

1. Start JDeveloper using the Oracle Fusion Applications Developer profile.
   Seed Data user interface tasks are run from within a JDeveloper Business Components project.

2. Create the ADF Business Components artifacts to represent the logical data model, including entity objects and view objects in the project. Create the relationships between the entity objects and view objects in accordance with the Applications Standards. Add the objects to an application module that will serve as the entry point for the Business Service Object. The application module also serves as the container for the Seed Data Framework Configuration meta-data.

3. Run the Seed Data Configuration wizard.
   The wizard is the graphical user interface tool used to configure Application Modules for Seed Data.
   To launch the Seed Data Configuration Wizard, right-click the application module name in the Application Navigator tree view, and select Seed Data > Configuration > Edit as shown in Figure 58-1.
Figure 58-1 Starting Seed Data Configuration

Note:

If the seed application module in use is derived from a subclass of OAApplicationModuleImpl, that subclass and its dependencies, if any, should be made available in compiled form using the Libraries/Classpath feature of JDeveloper. If this step is not done, extract and upload will fail.

The Seed Data Configuration wizard launches and displays the Driver Definition Panel.

This panel, shown in Figure 58-2, initially shows the available view objects in the root level of the application module data model that can serve as Driver view objects.

Figure 58-2 Seed Data Configuration Driver Panel

a. Select the Driver check box for those view objects to serve as the Driver for Seed Data Extract. More than one Driver can exist within an application module; however, only one Driver can be specified at Extract time.
b. Select the **Configure** radio button for the Driver view object to configure it during this session. Only one driver view object can be configured during a single editing session.

4. Click **Container Panel**.

The Container Panel, as shown in **Figure 58-3**, shows the contained and reference relationships of the Driver view object. Also shown are the underlying table names that will be extracted from and updated to during the Extract and Upload processes.

**Figure 58-3 Seed Data Configuration Container Panel**

![Seed Data Configuration Container Panel](image)

**View object relationships**: The tree displays the model of the Seed Data Configuration. Relationships between the view objects are displayed. The relationship is either contained or just a reference. View links between the view objects that are based on Associations marked explicitly as Composition Association are shown as contained. Seed Data operations will be performed for all the view objects that are identified as contained.

Foreign key relationships are called reference relationships. They are identified by the existence of a view link between view objects backed by non-composite entity associations, or no entity associations at all, or a join between two entity objects (the referred entity object is marked as **Reference** in the View Definition) and a List of Values on the **name** field.

**Tables Information**: The list shows the tables that are updated during Seed Data upload. A non-editable list of tables is generated based on the declared data model indicated by a Source column value of **Model**. The Type column indicates whether the table is just Referenced or Updated. These tables and the explicitly added tables with Type **Updated** will be frozen for (near) Zero-Downtime patching.

**Table Freeze** involves making the table in the Production edition read-only and creating a replacement table in the **Patch** edition. The list of seed tables that will be inserted, updated or referenced during seed data upload is provided as metadata to the patching utility. This is auto-generated by the Configuration...
wizard by inspecting the underlying entity objects and base tables for a given
driver view object.

If the entity objects of the configuration are not ADF Business Components-based,
or custom insert or updates exist (such as through PL/SQL code), the seed data
framework will not be able to accurately determine the set of tables participating
in the seed data upload. In such cases, the seed data framework will not be able to
accurately determine the set of tables to be frozen. The owner of the seed
application module must declare the additional tables in the container panel.
Failure to review and declare tables (such as by specifying a list that is incorrect or
incomplete) that are participating in seed data upload is likely to cause data
invalidation or corruption, as some tables will not be frozen and the Production
Instance will directly be aware of seed data upload changes that should be limited
to Patch Instance. Patch rollback also could leave orphan records in these cases.

Use the Add and Remove buttons to add or remove additional tables that are
affected from the list. The list of updated tables is for information only. This
information is used during patch application.

**Table Name**: This column shows the name of the table that is affected during seed
data upload.

**View Name**: This column shows the name of the ADF Business Components view
object where this table is declared. This column can also contain the name of the
Java class or PL/SQL procedure which would be used for seed data upload.

**Source**: Shows the source of this definition. The value of M is reserved to indicate
that the Table Name is derived from the ADF Business Components Model. You
can create your own definitions for additional table entries.

**Type**: This column shows the update type of the definition. A value of Updated
indicates that the definition table will be Updated at Seed Data Upload time. A
value of Referenced indicates that this is a Referenced table only, and will not be
updated by Seed Data Upload.

Click the **Surrogate Panel**.

Use the Surrogate Panel, shown in **Figure 58-4**, to declare surrogate attributes of
the view objects of the Seed Data Configuration.
The tree displays the model of the Seed Data Configuration. View objects can be selected in the tree to declare a surrogate attribute of the selected view object.

**Note:**

If you have an entity object with a Surrogate ID that is involved in a parent-child relationship, but you are not able to view or select that Surrogate ID on the Surrogate panel, check the data model and confirm that you have checked the composition check box for that association. See the "Creating and Configuring Associations" section of the *Developing Fusion Web Applications with Oracle Application Development Framework*.

**Surrogate Attribute – Available:** The left panel lists the attributes of the selected view object that can possibly be a surrogate attribute. The list will include primary key (PK) attributes that are of data type numeric.

**Surrogate Attribute:** The list on the right contains the attribute of the view object that has been identified as a Surrogate Attribute.

**Alternate Key:** Select the Alternate Key that will be used as the unique row identifier for the selected view object. The Alternate Key choice is based on the Alternate Keys available on the entity object of the selected view object. If the Primary Key has many attributes and one of them is being marked as a surrogate, all the other attributes in the Primary Key, except the one being marked as surrogate, must be included in the alternate key for that alternate key to be displayed in the list.

For Date Effective models, the above rule has been relaxed so that Effective Start Date, Effective End Date, Date Effective Sequence and Date Effective Flag are not required to be in the alternate key, even if they happen to be in the Primary Key to appear in the list.

To declare an attribute as a Surrogate Attribute:
a. Select the view object that contains the surrogate attribute. The left bottom panel will show the list of attributes that are potential surrogate attributes.

b. Select the Surrogate Attribute and click the shuttle button.

Click **Reference Panel**.

Use the Reference Panel, shown in Figure 58-5, to declare the Data Upload Mode for the Seed Data Configuration. Provide reference information on reference view objects.

**Figure 58-5  Seed Data Configuration Reference Panel**

The tree displays the model of the Seed Data Configuration. Reference view objects can be selected in the tree to declare information about the Seed Data Configuration of the referred view object. *All the external references must be specified for the patch to succeed.*

a. In the tree view, select the reference view object. The bottom left panel will contain a list of available application modules.

b. Select the application module that contains the Seed Data Configuration information for the reference view object.

c. Add a reference in the current application module to this application module by clicking the shuttle button.

**Note:**

If there are any Service Foreign Key LOVs defined in your model, review the different scenarios in Table 58-2 and verify that the steps listed for the scenario corresponding to your FK LOV are followed. Failure to follow the instructions in the document can cause problems during Seed Data Extract and Upload processes.
Table 58-2 Service Foreign Key LOV Scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>What must be done</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1: Single attribute Foreign Key and single attribute Alternate Key (Example: PersonVO has Deptno and Dname. Deptno is the foreign key ID, and Dname is the foreign alternate key.)</td>
<td>An LOV should be defined on the alternate key attribute, with the foreign key ID as a derived attribute.</td>
</tr>
</tbody>
</table>

1. Create PersonVO based on EmpEO and a reference DeptEO. The foreign alternate key (Dname) from the reference entity object is included in the EmpVO.
2. Define the LOV view object (DeptVVO).
3. Define a view accessor on EmpEO/PersonVO pointing to DeptVVO.
4. Define an LOV on the foreign alternate key (Dname) using the above view accessor, and configure the LOV to populate the foreign key id (Deptno) as the derived attribute.
5. Manually edit the VO.xml and add LBThrowOnMisMatch="true", such as:

```xml
<VObject xmlns="http://xmlns.oracle.com/bc4j" Name="SdLovrefReferringVO" Version="11.1.1.53.68" SelectList="SdLovrefReferringEO.ID, SdLovrefReferringEO.REFERRING_DATA, SdLovrefReferedEO.REFERRED_ALTATTR_2, SdLovrefReferedEO.REFERRED_ALTATTR_1, SdLovrefReferedEO.REFERRED_ID, SdLovrefReferringEO.REFERENCE_ID" FromList="SD_LOVREF_REFERRING SdLovrefReferringEO, SD_LOVREF_REFERRED SdLovrefReferedEO" BindingStyle="OracleName" CustomQuery="false" RowClass="oracle.apps.fnd.applcore.oaext.model.OAViewRowImp" ComponentClass="oracle.apps.fnd.applcore.oaext.model.OAViewObjectImp" PageIterMode="Full" LBThrowOnMisMatch="true" UseGlueCode="false" Where="SdLovrefReferringEO.REFERENCE_ID = SdLovrefReferedEO.REFERRED_ID(+)>
### Table 58-2  (Cont.) Service Foreign Key LOV Scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>What must be done</th>
</tr>
</thead>
<tbody>
<tr>
<td>#2: Single attribute Foreign Key and multiple attribute Alternate Key</td>
<td>Each alternate key attribute needs to have an LOV defined, and each LOV should have all the alternate key attributes as the driving attribute, and the foreign key ID as the derived attribute.</td>
</tr>
<tr>
<td>(Example: PersonVO has OrganizationId as foreign key ID, and OrganizationName +BusinessGroupName as the composite alternate key.)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. Create PersonVO based on EmpEO and a reference OrganizationEO and another reference BusinessGroupEO. The foreign alternate key (OrganizationName and BusinessGroupName) from the reference EOs are included in the PersonVO.</td>
</tr>
<tr>
<td></td>
<td>2. Define the LOV view object (OrganizationVVO).</td>
</tr>
<tr>
<td></td>
<td>3. Define a view accessor on EmpEO/PersonVO pointing to OrganizationVVO.</td>
</tr>
<tr>
<td></td>
<td>4. Define an LOV on each of the foreign alternate key attributes (OrganizationName and BusinessGroupName) using the above view accessor, and configure the LOV to populate the foreign key id (OrganizationId) as a derived attribute.</td>
</tr>
<tr>
<td></td>
<td>5. Modify the PersonVO.xml to make the LOVs driven by all the foreign alternate key attributes, such as:</td>
</tr>
<tr>
<td></td>
<td>&lt;ListBinding</td>
</tr>
<tr>
<td></td>
<td>Name=&quot;LOV_OrganizationName&quot;</td>
</tr>
<tr>
<td></td>
<td>ListVOName=&quot;OrganizationVA&quot;</td>
</tr>
<tr>
<td></td>
<td>ListRangeSize=&quot;-1&quot;</td>
</tr>
<tr>
<td></td>
<td>NullValueFlag=&quot;none&quot;</td>
</tr>
<tr>
<td></td>
<td>NullValueId=&quot;LOV_OrganizationName_LOVUIHints_NullValueId&quot;</td>
</tr>
<tr>
<td></td>
<td>MRUCount=&quot;0&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;AttrArray Name=&quot;AttrNames&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Item Value=&quot;OrganizationName&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Item Value=&quot;BusinessGroupName&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/AttrArray&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;AttrArray Name=&quot;DerivedAttrNames&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Item Value=&quot;OrganizationId&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/AttrArray&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;AttrArray Name=&quot;ListAttrNames&quot;&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Item Value=&quot;OrganizationName&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Item Value=&quot;BusinessGroupName&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;Item Value=&quot;OrganizationId&quot;/&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;/AttrArray&gt;</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>6. Manually edit the VO.xml and add LBThrowOnMisMatch=&quot;true&quot;</td>
</tr>
</tbody>
</table>

---

Using the Seed Data Loader in JDeveloper
### Table 58-2  (Cont.) Service Foreign Key LOV Scenarios

<table>
<thead>
<tr>
<th>Scenario</th>
<th>What must be done</th>
</tr>
</thead>
<tbody>
<tr>
<td>#3: Single attribute Foreign Key and multiple attribute Alternate Key and one of the alternate key attributes is another foreign key. (Example: PersonVO has BirthOfCountry as foreign key ID, and BirthOfCity as another foreign key ID. BirthOfCity has BirthOfCountry+CityName as a composite alternate key.)</td>
<td>The first foreign key (BirthOfCountry) must be resolved first, either based on Scenario #1 or #2. Then the second alternate key should filter by the first alternate key.</td>
</tr>
<tr>
<td></td>
<td>1. Define an LOV view object based on CountryEO.</td>
</tr>
<tr>
<td></td>
<td>2. Define an LOV view object based on CityEO. Define a view criteria to filter by CountryId.</td>
</tr>
<tr>
<td></td>
<td>3. Define a view accessor on EmpEO/PersonVO pointing to CountryVVO.</td>
</tr>
<tr>
<td></td>
<td>4. Define a view accessor on EmpEO/PersonVO pointing to CityVVO, and bind CountryId to BirthOfCountry.</td>
</tr>
<tr>
<td></td>
<td>5. Define an LOV on CountryName using CountryVVO view accessor, with BirthOfCountry as a derived attribute from CountryId from CountryVVO.</td>
</tr>
<tr>
<td></td>
<td>6. Define an LOV on CityName using CityVVO view accessor, with BirthOfCity as a derived attribute from CityId from CityVVO.</td>
</tr>
<tr>
<td></td>
<td>7. Manually edit the VO.xml and add LBThrowOnMisMatch=&quot;true&quot;</td>
</tr>
<tr>
<td>#4: Composite foreign key</td>
<td>Each foreign key ID will be dealt with individually. For example, the foreign key id is OrgId+SourceId, then orgId and SourceId should be resolved based on solution in #1 or #2 or #3 separately. Then a validator must be defined to make sure combination of OrgId and SourceId is valid. This has the assumption that each individual attribute are a primary key itself.</td>
</tr>
</tbody>
</table>
Note:

A UI-only LOV can be defined without a derived attribute. The Seed Data Framework normally would ignore such an LOV and no external reference metadata would be generated for it.

In some cases, it might be desirable to let the Seed Data Framework treat such an LOV as a regular LOV with derived attributes. To do this, you need a "fake" derived attribute defined on the LOV.

For example:

\[ \text{LinesStatusEO(LineStatusId, StatusCode)} \]
\[ \text{StatusEO(StatusCode)} \]

1. Create a join between LineStatusEO and StatusEO in LineStatusVO.
2. Mark StatusEO as Reference.
3. Include the StatusCode attribute from StatusEO into LineStatusVO.
   
   Note: The attribute would be named StatusCode1, since StatusCode from LineStatusEO is already a part of LineStatusVO.
4. Define the LOV on StatusCode with the derived attribute as StatusCode1.

58.2.3 How to Use the Seed Data Extract Manager

Use the Seed Data Extract Manager tool to extract seed data from the pre-configured application modules. The generated extract files are partitioned by the module owner. Data can be filtered during extract time to limit the number of data files generated.

Note:

If you change the name of a seed data file after Extract, you also must manually update any references to that file name. Otherwise, proper ordering for Upload during patching run time for the files will not work as expected. So long as the physical file names and the names in task references match, the patching utilities will sequence the seed data tasks correctly.

The Seed Data Extract is driven off the Driver view object, as defined in the application module Configuration. This Driver view object serves as the root of the extract, and any contained child objects are extracted as containing data. Only one Driver view object is active during the Extract process.

There are two methods of starting a Seed Data Extract process:

- From JDeveloper using the Seed Data Framework Console
- By using an external Command Line Interface, as shown in Using the Extract Seed Data Command Line Interface

**To launch the Seed Data Framework Console from JDeveloper:**

The Seed Data Console is the graphical user interface (GUI) tool used to run both Extract and Upload Seed Data tasks. The Console also can be used to view the underlying table data for the view objects of the application module data model.
Right-click the application module name in the Application Navigator tree view and select **Seed Data > Extract** to launch the Seed Data Framework Console, as shown in **Figure 58-6**.

**Figure 58-6 Launching the Seed Data Console**

If there are multiple database connections in the workspace, the Seed Data Console – Select Database dialog, shown in **Figure 58-7**, displays.

Although this is not the normal case, if you need to debug data on two different databases, this lets you choose which one to use.

**Figure 58-7 Select Seed Data Database**

Select from the list of database connections available to the project. The default selected database is the default connection set on the Project Properties, under Business Components.

Click **OK** to launch the Seed Data Console with the selected connection information.

The Seed Data Console main page, as shown in **Figure 58-8**, displays.
The tree view shows the selected application module name as the root node, and each of the configured Driver view objects as child nodes under the root. Only Driver view objects as configured by the Seed Data Configuration wizard will show in this view.

The right side of Console is the output area, where processing messages of Seed Data tasks are displayed.

Note:

The Seed Data Console does not specify the last applied version of the seed data file. Consequently, users always will see a warning message indicating incremental uploads have been turned off. This is harmless and can be ignored.

In the log, the warning will appear similar to:

```plaintext
[WARNING] 14:38:58 : -cfver parameter not passed, incremental uploads turned off
```

### 58.2.4 How to Use Seed Data Extract Processing

To start a Seed Data Extract operation, right-click the Driver view object from which you want to extract and select **Extract Seed Data** from the context menu, as shown in Figure 58-9.

The Select Extract Path dialog, shown in Figure 58-10, displays.
58.2.4.1 Understand Extract Taxonomy Partition Selection Dialogs

Seed Data Extract uses selection dialogs for Taxonomy partitions. The information here covers the functional design and the various dialogs and partitions you will encounter.

**Functional Design**

When starting the Extract process, if a Taxonomy Partition Attribute is found in the Driver view object, Extract will show the Taxonomy Partition selection dialogs. See "Determining the Taxonomy Partitioning Attribute" for steps taken by Extract to find the Taxonomy partition attribute, if any.

**Taxonomy Products Dialog**

The Select Application Taxonomy Product Modules, as shown in Figure 58-11, will display a fixed list of all the Taxonomy products enabled for use with the Seed Data Framework. If the list of products for which seed data extract needs to happen is known beforehand, users can pick them from the list of available products. Optionally, click Filter to filter the list of products to show only those products and Logical Business Area (LBA) modules for which records exist in the selected view object. Occasionally, when there are a large number of records involved, the it might take a while to filter the list of products.
If any Products or LBAs are selected that are not actually available in the view object, no attempt is made to extract for that partition. This is done by applying implicit Partition criteria, binding for each selected partition, and verifying at least one row exists for the Partition row set before attempting to extract.

For internal development test cases and debugging for Extract, you will need to know which partitions are available. To do so, you have to select the ModuleIds from the driver and discover which Product that ModuleId equates to from the Taxonomy table. If the driver view object moduleId is an LBA, you also will need to know under which Product that LBA falls; this involves selecting from the Taxonomy hierarchy table.

An alternate method is to just select all the Products and all the LBAs when prompted by the dialog. Then extract files are created for only those modules that actually exist. This will be a little slower, since building the complete LBA list from all Products can take a few seconds or longer. Then each selected partition is bound to determine the availability. This will not be the typical Applications use case, as developers will know which Products to select. It is necessary in a development case in which the Products are not known.

**LBA Taxonomy Partitions**

After selecting the Products, the second dialog, Select Application Taxonomy LBA Modules, may or may not be shown. If the selected Products were parents of explicitly available seed-enabled LBA modules, all the available LBAs for the selected Products are shown. Again, only LBA module types found in the driver view object rows, and set as Seed Enabled in the Taxonomy Service, are shown in the list.

If the selected Products were not parents, or there are no LBAs available, no LBA selection dialog is shown, and Extract is started for all the selected Products.

For the LBA selection dialog, the full LBA taxonomy path is shown, showing any parents of subLBAs, and the parent Product short name. Again, the user selects the desired LBA values by shuttling values to Selected.

**Figure 58-12** shows the Select LBA Taxonomy Partitions dialog showing all the available LBAs found for the previously selected Receivables (AR), Payments (IBY), and Human Resources (PER) Products.
Extract Processing

After selecting LBAs, Extract proceeds to extract selected LBAs and selected base (non-parent) Product types, if any. Each Product and LBA type name value corresponds to a unique Extract file. Each Extract file will contain all rows containing the Taxonomy Partition Attribute (ModuleId or ApplicationId) value corresponding to each selected type.

Extract files are placed in folders according to the taxonomy path, starting at the Product short name, and including any LBAs and subLBAs as subdirectories. All the Extract file names follow the same pattern: <driver view object name>SD.xml.

Figure 58-13 shows a sample output selecting the three Receivable (AR) LBAs, creditCardErrors, customerProfileClasses, and miscellaneousReceipts, and General Ledger (AR) and Opportunity Management (MOO) Applications. The applications, AR and MOO, were explicitly defined in the entity, and contained no child LBAs.

Determining the Taxonomy Partitioning Attribute

The Taxonomy Partition Attribute on the Seed Driver view object is determined in the following manner, in order of precedence:

1. Attribute serving as ViewLink to ApplTaxonomySeedDataPVO ModuleId or AlternativeId attributes, or ApplicationPVO ApplicationId attribute.

2. Attribute serving as LOV to ApplTaxonomySeedDataPVO UserModuleName or AlternativeName attributes.
3. Hard-coded ModuleId attribute

4. Hard-coded ApplicationId attribute

Single File Implicit Partitioning

If no driver view object attributes pass above, single file implicit partitioning will be used and all rows will be extracted to a single extract file. In this case, no taxonomy partitioning dialogs will be shown.

The taxonomy owner for the corresponding file path is determined from the ApplicationModule package name.

For example, extracting from `oracle.apps.fnd.lookups.service.FndLookupsAM` will create the extract file in the FND folder under the user-specified extract root path.

Extract Manager Support for Seed Data File Dependencies

The Extract function adds metadata of the file on which the current file is dependent.

For example, the metadata shown in Example 58-1 would be added using adxml comments for the case where the current file is dependent on it.

Example 58-1 Sample Metadata Using ADXML Comments

```xml
<!-- <run_after_tasks> -->
<!-- adxml: <task_reference te="CZ" subdir="CZ"
file_basename="ReferredSD.xml" identifier="UPLOAD"/> -->
<!-- adxml: </run_after_tasks> -->
```

adxml are the comments added to the extract file at the beginning of the file. They are the same as normal xml comments except they have `adxml:` prepended to the commented text.

Extract gathers this information by using the Reference view object from Reference application module. Reference application module is configured by the user in the Reference panel of Seed Data Configuration. See Figure 58-5.

This information is used by the Patching Utility to create the order in which the files need to be uploaded so that the Reference data is available before the Referring data is loaded.

Static File Dependencies

For the cases where dynamically finding the file dependencies is not possible, or dynamic dependencies are not complete, you can set static file dependencies using an application module custom property for a view object instance, as shown in Table 58-3.

All files extracted from that view object would have the dependencies stamped in adxml comments.
Table 58-3  Application Module Custom Property for a View Object Instance

<table>
<thead>
<tr>
<th>Application Module Custom Property Name</th>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD_DEPENDENT_FILES_&lt;view object instance name&gt;</td>
<td>&lt;product&gt;:&lt;path&gt;:&lt;filename&gt;, &lt;product&gt;:&lt;path&gt;:&lt;filename&gt;,..</td>
<td>Property: SD_DEPENDENT_FILES_TimeDefinition1</td>
</tr>
<tr>
<td></td>
<td>The static dependent file location consists of three parts separated by colons:</td>
<td>Value: HCM:HCM/Per:LookupsSD.xml,FND:FND:ValueSetSD.xml</td>
</tr>
<tr>
<td></td>
<td>•  product: Product short name.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>•  path: Relative path from the product folder where the file exists.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>•  filename: name of the seed xml file that contains the referenced data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Multiple static external references should be separated by a comma.</td>
<td></td>
</tr>
</tbody>
</table>

Turning Off Dynamic File Dependency Generation

You can turn off the dynamic file dependency generation by defining the custom property shown in Table 58-4 on the application module.

Table 58-4  Property to Turn Off Dynamic File Dependency

<table>
<thead>
<tr>
<th>Application Module Custom Property Name</th>
<th>Value</th>
<th>Example</th>
</tr>
</thead>
</table>
| SD_NO_DYNAMIC_EXT_REFS_<view object instance name> | true  | Property: SD_NO_DYNAMIC_EXT_REFS_FndObjects
|                                          |       | Value: true |

Output Log Level

The Log Level for the Seed Data tasks is used to increase or decrease the amount and type of log messages generated during processing. The default Log Level is set to INFO. This will display generated severe errors, warnings, and informational processing messages. To limit the number of log messages generated, set the log level higher, to Severe or Warning. To see more processing messages generated for debugging purposes, set the Log Level to a lower level. Set the Log Level to FINEST to see the most processing messages generated. These messages will generally only be useful to developers.

58.2.4.2 Using the Extract Seed Data Command Line Interface

The Seed Data Extract process also can be initiated externally from JDeveloper using the Command Line Interface (CLI). Seed Data Extract command line parameters can be passed using one of two methods:

- Directly on the Java command line
- By using a command property file that is singularly passed on the command line
Extract Seed Data Java Command Line Syntax

If you are using a Linux operating system, use these commands:

```java
java -cp $jdev_install/jdeveloper/jdev/oaext/lib/oracle.apps.fnd.applseed-rt.jar:
$jdev_install/oracle_common/modules/oracle.odi_11.1.1/ojdl.jar
oracle.apps.fnd.applseed.rt.extract.Extract
   -dburl <database connect url string without username and password>
   -dbuser <database user>
   -AM <fully qualified path to seed configured AM>
   -VO <seed Driver view object instance name>
   -dburl <database connect URL string without username and password>
   -dbuser <database user>
   -AM <fully qualified path to seed configured AM>
   -VO <seed Driver view object instance name>

[-ExtractRootPath <output path location>]
[-PartitionKeyIds <Taxonomy ModuleId values> | -PartitionKeyNames <Taxonomy short name values>]
[-PartitionKeyIds <partition key id values>]
[-PartitionKeyNames <partition key names>]
[-log <Used to give the log Level (SEVERE to FINEST)>]
[-loglevel <Used to give the location of the log file>]
[-entid <enterprise id>]
```

If you are using a Windows operating system, use these commands:

```java
java -cp %jdev_install%\jdeveloper\jdev\oaext\lib\oracle.apps.fnd.applseed-rt.jar;
%jdev_install%\oracle_common\modules\oracle.odi_11.1.1\ojdl.jar
oracle.apps.fnd.applseed.rt.extract.Extract
   -dburl <database connect url string without username and password>
   -dbuser <database user>
   -AM <fully qualified path to seed configured AM>
   -VO <seed Driver view object instance name>
   -dburl <database connect URL string without username and password>
   -dbuser <database user>
   -AM <fully qualified path to seed configured AM>
   -VO <seed Driver view object instance name>

[-ExtractRootPath <output path location>]
[-PartitionKeyIds <Taxonomy ModuleId values> | -PartitionKeyNames <Taxonomy short name values>]
[-PartitionKeyIds <partition key id values>]
[-PartitionKeyNames <partition key names>]
[-log <Used to give the log Level (SEVERE to FINEST)>]
[-loglevel <Used to give the location of the log file>]
[-entid <enterprise id>]
```

Command Property File

The command property file is an external file that contains the command line properties in standard Java Properties format for each of the required and optional Extract command line properties. The format can be name=value, or name:value.

If you are using a Linux operating system, use this command:

```java
java -cp $jdev_install/jdeveloper/jdev/oaext/lib/oracle.apps.fnd.applseed-rt.jar:
$jdev_install/oracle_common/modules/oracle.odi_11.1.1/ojdl.jar
oracle.apps.fnd.applseed.rt.extract.Extract [command property file]
```

If you are using a Windows operating system, use this command:
java -cp "$jdev_install%/\jdeveloper\jdeveloper\oaext\lib\oracle.apps.fnd.appseed-rt.jar; $jdev_install%/oracle_common\modules\oracle.odi_11.1.1.1\ojdl.jar oracle.apps.fnd.appseed.rt.extract.Extract [command property file]

**Seed Data Extract Command Line Parameters**

The available Seed Data Extract parameters are listed in Table 58-5.

**Table 58-5  Available Seed Data Parameters**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Required?</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>dburl</td>
<td>database connection URL in JDBC format without username and password</td>
<td>Yes</td>
<td>jdbc:oracle:thin:@fully_qualified_server_name:1991:database_name</td>
</tr>
<tr>
<td>dbuser</td>
<td>database user to be used for extract</td>
<td>Yes</td>
<td>fusion</td>
</tr>
<tr>
<td>AM</td>
<td>Application module name, fully package qualified</td>
<td>Yes</td>
<td>oracle.apps.fnd.applcore.flex.dff.descriptiveFlexfieldService.</td>
</tr>
<tr>
<td>V0</td>
<td>Driver ViewObject instance name</td>
<td>Yes</td>
<td>DescriptiveFlexfield</td>
</tr>
<tr>
<td>ExtractRootPath</td>
<td>Path to extract seed data files</td>
<td>No</td>
<td>/home/seed/data</td>
</tr>
<tr>
<td>PartitionKeyId</td>
<td>Comma delimited Taxonomy Id values to extract, either ModuleId or ApplicationId, depending on partition strategy.</td>
<td>No</td>
<td>Taxonomy Module Ids: 4F1F0DFC58F87DB4E04044981FC6 2F46, 47110F64AC8F08E2E040449823C60 DB6 Application Ids: 250, 667, 10047</td>
</tr>
<tr>
<td>PartitionKeyName</td>
<td>Comma delimited Taxonomy name values to extract, either Product codes, or LBA names, or combination thereof</td>
<td>No</td>
<td>FND, HCM, invoices, receivables, cashManagement</td>
</tr>
<tr>
<td>loglevel</td>
<td>The Log Level (SEVERE to FINEST).</td>
<td>No</td>
<td>-loglevel FINE</td>
</tr>
<tr>
<td>log</td>
<td>The location of the log file.</td>
<td>No</td>
<td>-log /home/seed/ex.log</td>
</tr>
<tr>
<td>Entid</td>
<td>Enterprise Id numeric value.</td>
<td>No</td>
<td>-entid 1</td>
</tr>
</tbody>
</table>

**Note:**

The database password would be prompted. To avoid prompting, pipe it on the command line. The password must be piped in when output is redirected. For example, to pipe a password from the $FUSION_PASS environment variable to the Extract command line:

```
java Extract cmdline... <<! $FUSION_PASS
```
**PartitionKey<Ids|Names> Properties**

The PartitionKey properties drive how the seed data extract derives the data file partitions, which is the number of files generated. Each partition key will equate to a single extracted seed file, with all the rows that are owned by that particular module being extracted to its seed file.

You can use either the PartitionKeyId or the PartitionKeyName property, or a combination of both, to supply to the extract each of the unique file partitions that will be created. If no PartitionKey properties are specified, the default behavior is to extract all file partitions found from the driver view object, and all rows extracted to each corresponding seed data file.

You should use one of the PartitionKey parameters to limit the amount of files generated. Otherwise, the expected partitions will need to be determined from executing the driver view object query and perform a complete table scan of all rows. For very large tables with many thousands of rows, this could be a potentially large performance hit, and, depending on the complexity of the view object query and its joins, could take several minutes to hours to determine.

**Property File Comments**

In the command property file, any lines beginning with a pound sign (#) will be considered comments, and not processed in any way by the Extract tool.

You also can comment out specific entries in the multi-value comma delimited properties. For example:

```
PartitionKeyName = FND, HCM, #FCM, GL, AM
```

This will ignore the FCM value entry, but keep others intact. A sample PartitionKeyName command line option is shown in Example 58-2.

**Example 58-3** shows the contents of the sample Seed Extract Command Property File, located at `/home/extract.properties`:

**Sample command line showing command line parameter to `/home/extract.properties` command property file.**

```
java -cp $jdev_install/jdeveloper/jdev/oaext/lib/oracle.apps.fnd.applseed-rt.jar:
$jdev_install/oracle_common/modules/oracle.odl_11.1.1/ojdl.jar
oracle.apps.fnd.applseed.rt.extract.Extract /home/extract.properties
```

If you are using a Windows operating system, follow this example:

```
#Sample extract property file, comment line
AM
=oracle.apps.fnd.applcore.flex.dff.descriptiveFlexfieldService.applicationModule.Desc
riptiveFlexfieldServiceAM
dburl = jdbc:oracle:thin:fusion/fusion@fully_qualified_server_name:1991:database_name
dbuser = fusion
VO = DescriptiveFlexfield
ExtractRootPath = C:\seed\data
PartitionKeyName = FND,HCM
```

**Sample command line showing command line parameter to C:\extract.properties command property file.**

```
java -cp %jdev_install%\jdeveloper\jdev\oaext\lib\oracle.apps.fnd.applseed-rt.jar;
%jdev_install%\oracle_common\modules\oracle.odl_11.1.1\ojdl.jar
oracle.apps.fnd.applseed.rt.extract.Extract C:\extract.properties
```
**Example 58-2 Sample Command Line**

If you are using a Linux operating system, follow this example:

```java
java -cp ${jdev_install}/jdeveloper/jdev/oaext/lib/oracle.apps.fnd.applseed-rt.jar:
  ${jdev_install}/oracle_common/modules/oracle.odl_11.1.1/ojdl.jar
oracle.apps.fnd.applseed.rt.extract.Extract
  -dburl jdbc:oracle:thin:@fully_qualified_server_name:1991:database_name
  -dbuser fusion
  -AM oracle.apps.fnd.applcore.flex.dff.descriptiveFlexfieldService.applicationModule.Descr
  -VO DescriptiveFlexfield
  -ExtractRootPath /home/seed/data
  -PartitionKeyNames HCM, invoices, cashManagement
  -loglevel FINER
```

If you are using a Windows operating system, follow this example:

```java
java -cp %jdev_install%\jdeveloper\jdev\oaext\lib\oracle.apps.fnd.applseed-rt.jar;
%jdev_install%\oracle_common\modules\oracle.odl_11.1.1\ojdl.jar
oracle.apps.fnd.applseed.rt.extract.Extract
  -dburl jdbc:oracle:thin:@fully_qualified_server_name:1991:database_name
  -dbuser fusion
  -AM oracle.apps.fnd.applcore.flex.dff.descriptiveFlexfieldService.applicationModule.Descr
  -VO DescriptiveFlexfield
  -ExtractRootPath C:\seed\data
  -PartitionKeyNames HCM, invoices, cashManagement
  -loglevel FINER
```

**Example 58-3 Sample Command Property File**

If you are using a Linux operating system, follow this example:

```properties
#Sample extract property file, comment line
AM = oracle.apps.fnd.applcore.flex.dff.descriptiveFlexfieldService.applicationModule.Descr
/dburl = jdbc:oracle:thin:fusion/fusion@fully_qualified_server_name:1991:database_name
/dbuser = fusion
/VO = DescriptiveFlexfield
/ExtractRootPath = /home/seed/data
/PartitionKeyNames = FND,HCM
```

**58.2.5 How to Use the Seed Data Upload Manager**

To launch the Seed Data Framework Console, refer to "To launch the Seed Data Framework Console from JDeveloper:" but select **Seed Data > Upload**.

When you right-click a Driver view object in the tree list, the menu shown in **Figure 58-14** displays so you can select the **Load Seed Data** option.
In addition to the Load Seed Data option, discussed in Uploading Seed Data, three other options are available:

**Set Log Level**

Set how much information you want written to the log file. The least amount of information will be if this is set to Severe, and the largest amount of information will be if this is set to Finest. The default setting is Info, which will log Information, Warning and Severe messages.

**Clean Mode**

The default setting is Disabled. If Clean Mode is Enabled, it basically deletes all the existing records before proceeding to upload the given file. This option is exposed both through the command line interface for upload (-clean) and here.

---

**Caution:**

*This option should be used extremely carefully as it might lead to irreversible data loss.*

---

**Customization Mode**

The default setting is Do Not Preserve.

Seed Data Loader always sets the last_updated_by and created_by to zero when it inserts new records, and it always sets the last_updated_by to zero when it updates existing records.

Customers who have customized some of the Seed Data records are expected to set last_updated_by to a non-zero value.

*By default when using the Seed Data Console, customizations are not preserved. They are overwritten.*

However, on the command line, which is primarily intended for use at the customer site and for automated uploads, customizations are preserved by default. To override this, use the -nocust option.

### 58.2.5.1 Upload Seed Data

To upload Seed Data, right-click the desired Driver view object from the Console view object tree list. Select Load Seed Data.
**Note:**

When you upload files, if the row already exists in the database and if it has been customized (last_updated_by <> 0), a "Skipped" message will be placed in the log and the row will not be updated. To correct this, change last_updated_by to 0 in both the database and the file before uploading.

The Select a file to load dialog, shown in Figure 58-15, displays.

**Figure 58-15 Selecting a File to Load**

![Select a file to load dialog](image)

Select either an XML file or its corresponding translation xmliff file to initiate a National Language Support (NLS) upload.

Click **OK** to begin the Upload process on the selected file.

**Upload Output**

When the Seed Data upload finishes, the processing messages are shown in the output tabbed view window, in the tab corresponding to the view object against which the upload was run. See Figure 58-16.
You may see warning messages about columns being not updateable. Review these messages to determine if you can ignore them for your specific case.

58.2.5.1.1 How to Upload Seed Data Using the Command Line Interface

The Seed Data Upload process can be initiated externally from JDeveloper using the Command Line Interface (CLI).

To run the command line version of the Seed Data Loader from within an ADE view, ensure the JDEV_HOME environment variable, shown in Table 58-6, is set to a valid JDeveloper installation directory. Include the jdeveloper sub folder if you are using JDeveloper integrated with WebLogic Server.

### Table 58-6 Environment Variables for the Seed Data Upload CLI

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Required?</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDEV_HOME</td>
<td>No</td>
<td>Should point to the full path to the JDeveloper installation.</td>
</tr>
<tr>
<td>APPLSEED_CLASS_PATH</td>
<td>No</td>
<td>Used to add additional classpath entries to the loader (in addition to the regular CLASSPATH setting which might not be modifiable in certain circumstances). This parameter is expected to include folders where Apps EAR archives are available in an exploded format (typically the product family level deploy folders). If the same ADF library is available from multiple locations, only one will be added to the classpath.</td>
</tr>
</tbody>
</table>
Table 58-6 (Cont.) Environment Variables for the Seed Data Upload CLI

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Required?</th>
<th>Purpose</th>
</tr>
</thead>
</table>
| APPLSEED_TS_CLASSPATH | No        | Used to add additional classpath entries to the loader (in addition to the regular CLASSPATH setting which might not be modifiable in certain circumstances). This parameter is expected to include folders where techstack libraries are made available. On a provisioned system, it should include at least these directories:  
  fmw1/atgpf/atgpf/modules/  
  oracle.applcore.model_11.1.1  
  fmw1/oracle_common/modules  
  fmw1/atgpf/modules  
  Note: The primary difference between APPLSEED_CLASSPATH and APPLSEED_TS_CLASSPATH is that the JAR files coming from APPLSEED_CLASSPATH are put through a unique filter that filters unique JAR files by file name. This is possible because ADF libraries follow a naming convention. However, there is no common naming convention across techstack libraries (there could be a util.jar from JDBC and another from the XML parser). Therefore, the JAR files coming from APPLSEED_TS_CLASSPATH are not filtered by filename. |
| APPLSEED_CLASSPATH_FILE | No        | Used to specify a file path, each line of which will be treated as a classpath entry similar to the entries in APPLSEED_CLASSPATH. |
| APPLSEED_TS_CLASSPATH_FILE | No        | Used to specify a file path, each line of which will be treated as a classpath entry similar to the entries in APPLSEED_TS_CLASSPATH. |

Command Line Syntax

Example 58-4 shows a sample Seed Data upload command entered on the CLI, and Example 58-5 shows a sample command line command for a Key Flexfield application module.

Example 58-4 Sample Seed Data Upload Command on the CLI

```java
java -cp $jdev_install/jdev/oaext/lib/key_ff_app_module_name.jar  
key_ff_app_module_name.loader.Loader  
-am <fully qualified application module name>  
-dburl <database url in jdbc format without username and password>  
-dbuser <database user>  
-file <complete path to the data file>  
[-config <bc4j config name>] // Example:  
  KeyFlexfieldServiceAMLocal, KeyFlexfieldServiceAMShared, KeyFlexfieldServiceAMTest  
[-atomic] // If used, the loader will load all the records or none of them (basically will stop after the first failure)  
[-nls] // Used to indicate to the loader that it should treat the given file as an xliff
```

Using the Seed Data Loader in JDeveloper

58-30 Oracle Fusion Applications Developer's Guide
[-clean]  // Cleans up any existing records before starting to upload from the file (use with caution)
[-loglevel]  // Used to give the log Level (SEVERE to FINEST)
[-log]  // Used to give the location of the log file
[-commitsize]  // This parameter is optional. The default value of commitsize is 1; the maximum allowed value is 9999. The commitsize parameter is used to decide the frequency for committing the seed loader transaction. For the default case with commitsize at 1, the transaction is committed after every 1 records, or after all the records are processed, whichever comes first. Note that the commits are only at the top level entity (and not in middle of child records), so commit may not occur exactly every N records, but at the top level entity after N records (top level and child records together) have been processed.

// Note: The previous name for the commitsize parameter was batchsize. However, this was getting confused with the JDBC batch value, and hence, it was decided to rename it to commitsize.
[-entid <enterprise id>]  // load data only for the given enterprise

Note:
The database password would be prompted. To avoid prompting, pipe it on the command line. The password must be piped in when output is redirected. For example, to pipe a password from the $FUSION_PASS environment variable to the Loader command line:

```
java Loader cmdline... <<! $FUSION_PASS
```

### Example 58-5  Sample Command Line Invocation for a KFF Application Module

If you are using a Linux operating system; follow this example:

```
java -cp $jdev_install/jdeveloper/jdev/oaext/lib/key_ff_app_module_name.jar:
$jdev_install/oracle_common/modules/oracle.odl_11.1.1/ojdl.jar
key_ff_app_module_name.loader.Loader
```

If you are using a Windows operating system; follow this example:

```
java -cp %jdev_install%\jdeveloper\jdev\oaext\lib\key_ff_app_module_name.jar;
%jdev_install%\oracle_common\modules\oracle.odl_11.1.1\ojdl.jarkey_ff_app_module_name.loader.Loader
```

#### 58.2.5.1.2 How to Invoke Seed Loader Modes

Three modes can be invoked when running a Seed Data Upload from the command line.

- **-clean**: This flag’s primary purpose is to make the set of records in the database reflect exactly what is delivered in the seed data file. If the set of records in the database becomes out of sync with what is delivered in the file, the -clean flag can be used to load the seed data file and make sure the database only has those records delivered from the file. This is turned off by default.

- **-atomic**: This flag’s primary purpose is to let the seed framework ensure that all the records in a given file are loaded. If all records are not loaded, none will be loaded. If even one record fails, the loading is stopped and all the previously loaded records are rolled back. That is, a single commit is issued towards the end of the file. This is turned off by default and a commit is done after every top level record.
- **nocust**: This flag's primary purpose is to let the seed framework know that it should not preserve any customizations done by the customer in the database version of the records. Data from the file is expected to overwrite anything that is in the database. This is turned off by default.

**Important Points**

- The -clean and -nocust flags are targeted for internal Oracle development. Given their potential to cause irrecoverable data loss, they are not recommended for production environments.

- The database password would be prompted. The password can be piped in to avoid prompting. The password must be piped in if output is redirected.

- A failure during the cleanup stage does not prevent the subsequent loading of the records from the file.

- The -clean, -nocust and -atomic flags can be used independently of each other.

- What is deleted when -clean is used?
  - The clean mode deletes all the records that satisfy a particular condition. In normal use cases, this is the partitioning criteria (ModuleId or ApplicationId) used when the data was extracted. The Seed data file has metadata about the partitioning attribute (ModuleId, ApplicationId or any other partitioning scheme) and the exact values for those attributes used during extract.

**Example**

Using the Messages model from the Oracle Fusion Middleware Extensions for Applications, when PER/SomePerLba developers extract their messages, they will receive a set of Messages owned by that LBA. The seed data file captures this metadata that is used during -clean to determine the set of records to remove. In this case, all the messages owned by PER/SomePerLba will be removed.

- The set of records deleted does not depend on the set of records in the file. It is only driven by the partitioning metadata stored inside the file. The loader would not have even read the records in the file by the time the cleanup is made.

**Example**

If PER/SomePerLba owns 10 messages in the database and an incoming MessageSD.xml brings in only six messages, which might be the same as or different from those in the database, the 10 messages in the database are all removed and the ones from the file are all loaded. If the loading succeeds, the database should have only six records owned by PER/SomePerLba - exactly what the seed data file brought in.

- Customizations and -clean

  The -clean mode currently does not preserve customizations. The seed framework identifies customized records by a non-zero last_updated_by. Such records are not updated by the seed framework and a warning is issued that the record has been customized in the database. The clean mode, however, does not currently honor this principle and deletes even the customized records.

- When are deletes committed?
The cleanup and the subsequent loading run in a single transaction. Therefore, a failure during the loading will roll back the deletes. Because the seed framework - by default - commits after every top level record, the cleanup - by default - is committed or rolled back along with the very first top level record. If subsequent top level records fail, they all will be left out of the database. Therefore, only a subset of the records that came in the file will remain. This behavior changes when using the -atomic flag. This flag essentially says “make sure all the records, or none of the records, in the file are loaded.” In this case, the cleanup and the loading are committed only once towards the end of the file. Even if one record fails, the deletes and any intervening loads are rolled back.

58.2.6 How to Share Application Modules

In some cases, Application Modules developed and owned by one team might have to be shared by some other team. In such cases, the team that developed the application module owns the module and the data-model associated with it, but the team that is consuming it owns the seed data and the relevant extract files.

In Figure 58-17, the team that is developing the application module packages the module into an ADF Business Components library.

The consuming team includes these libraries in the project using the Libraries/Classpath feature of JDeveloper. Once imported, developers can use these libraries to extract and upload seed data, but cannot edit the Seed Data configuration.

**Figure 58-17 Including Shared Libraries**

The imported Business Components can be seen using the Business Components Import feature of JDeveloper. Once imported, developers can use these libraries to extract and upload seed data, but cannot edit the Seed Data configuration.
Since Application Modules shared using an ADF Business Components library are not shown by JDeveloper in the Application Navigator, users must right-click the Business Components project into which they imported the shared libraries. The Seed Data menus would also be available on any Business Components project node in the Applications Navigator tree of JDeveloper.
Clicking any of the Seed Data Framework menu items will display a tree structure showing all the Seed Data Framework-enabled Application Modules (those that have at least one Seed Data driver view object configured within them) available to that project.

**Figure 58-20  Selecting an Application Module**

Users can choose an available application module and click **OK** to display the familiar Seed Data console to perform and extract or upload activities.

### 58.2.7 How to Update Seed Data

The Seed Data Loader supports incremental updates and Java Database Connectivity-based National Language Support updates. These are summarized in Table 58-7.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Applicable to</th>
<th>Default Setting</th>
<th>Design time Control</th>
<th>Runtime Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental Updates</td>
<td>Only US language seed data files</td>
<td>On</td>
<td>Disabled by setting SD_INCR_MIDTIER_&lt;ViewDefinitionName&gt; to false.</td>
<td>Disabled by using optional command line parameter -noincr</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Disabled by setting APPLSEED_NO_INCREMENTAL to true</td>
<td>- Disabled by setting APPLSEED_NLS_USE_ADF to true</td>
</tr>
<tr>
<td>JDBC Mode</td>
<td>Only translation seed data files</td>
<td>On for newly extracted files</td>
<td>Disabled by setting SD_NLS_USE_ADF_&lt;ViewDefinitionName&gt; to true</td>
<td>Disabled by setting APPLSEED_NLS_USE_ADF to true</td>
</tr>
</tbody>
</table>
58.2.7.1 Using Incremental Updates

The Seed Data Loader always updates all the records from the seed data file, even if those records are already present in the target database. The Seed Data Loader only updates the records from the seed data file that have changed in the target database. For certain kinds of requirements where it is desirable to have the seed data loaded as a set, and therefore incremental updates are not desirable, the Seed Data Framework provides an optional feature that allows product teams to disable this feature.

During extract, the Seed Data Framework computes MD5 checksums on the source record by taking all the fields that make up the record (excluding key attributes, history columns and non persistent fields). This checksum is embedded in the seed data file as additional record level metadata. At load time, if the record is found existing in the target database, the same algorithm is used to compute the checksum on the target record. By comparing the two checksum values, the seed data loader identifies if the record has undergone a change and needs an update. Updates are triggered only for those records for which the checksums do not match.

This incremental update feature is applicable to only US language Seed Data SD XML files and is enabled by default. For translated seed data, there is no provision to do incremental updates.

To disable this feature for a particular seed data driver view object and all its child view objects, the SD_INCR_MIDTIER_<ViewDefinitionName> property should be added to the application module.

Set the value of this property to false to turn off incremental updates.

SD_INCR_MIDTIER_ValueSetVO false

Note that the incremental updates feature requires new metadata, in the form of checksums, to be embedded into the seed data file. Consequently, only newer seed data files can take advantage of this feature. Note that the incremental updates feature requires new metadata in the form of checksums to be embedded into the seed data file. Consequently, only newer seed data files can take advantage of this feature. With older seed data files, the seed data loader shall continue to update all the records, even if the application module has the feature enabled using the SD_INCR_MIDTIER property.

As a debugging aid, the seed data loader also allows for incremental updates to be conditionally turned off at run time. Use the -noincr optional command line parameter to the loader or set the APPLSEED_NO_INCREMENTAL environment variable to true.

58.2.7.2 Implementing Java Database Connectivity-based National Language Support Updates

The Seed Data Loader loads seed data using the ADF components developed by product teams. Consequently, all the business rules, validation logic, and custom code built into Oracle Fusion Applications ADF components also are invoked. Translation seed data delivery usually involves only simple updates to existing records. Using plain JDBC calls, instead of ADF, to update existing records allows for much faster loads of translated seed data.

Newer translation seed data files have additional metadata that signal the loader to use the JDBC mode wherever possible. This additional metadata includes SQL fragments that the loader uses at load time to trigger JDBC updates, instead of the well-known ADF-based loads. This JDBC mode is designed to achieve functional
parity with the existing ADF mode, and is the preferred way to deliver translation seed data.

It is enabled by default for newly-created translation seed data files, except date effective translation seed data, which rely on additional logic built into the ADF. It should be noted that US language (XML) seed data files continue to be loaded through ADF. The JDBC mode is tailored only for translation (XML) seed data files. If certain kinds of translated seed data need to be always loaded using ADF, it is possible to do so using one of these ways:

- **Design time** - Define a custom property `SD_NLS_USE_ADF_<ViewDefinitionName>` and set its value to `true`.
- **Runtime** - Define an environment variable `APPLSEED_NLS_USE_ADF` and set its value to `true`.

### 58.3 Translating Seed Data

The Seed Data Framework will handle localized data stored in translation tables in a consistent manner. Translatable attribute data will be extracted separately for only the US base language into Ora-XLIFF compliant format files.

Translation teams will translate the base US language XLIFF into the various language translation files, one file for each supported language. The files are to be stored in a separate language folder, named for its language code.

At upload time, Seed Data Loader will recognize an incoming translation language XLIFF file automatically, and perform the necessary updates to the language tables.

#### 58.3.1 How to Extract Translation Data

Ora-XLIFF format files are automatically generated for those data models which have translatable data. Only the US language data is extracted. The XLIFF file is created in a language sub-folder, named US, and the file name is named the same as the base Seed Data SD XML file, with an extension of .xlf.

**58.3.1.1 Treating Seed Data Base XML and Language XLIFF as a Single Entity**

The seed data base and XLIFF files must be treated as a single entity. Any changes to either base or translated attributes that would require a re-extract, would necessitate that both the re-extracted base XML and US XLIFF files be delivered as units. There is metadata in the files to ensure that the base and XLIFF files match and were extracted together.

#### 58.3.2 How to Process Seed Data Translations

Translation teams will create the translation XLIFF files starting from the initial US file. Each supported language will have its own XLIFF file. The files will be stored in a separate language sub-folder, named for the language code. The files will be named the same as the base XML name, with the .xlf extension.

#### 58.3.3 How to Load Translation Seed Data

At upload time, the Seed Data Loader will recognize an incoming translation language XLIFF file automatically, and perform the necessary updates to the language tables, based on the target language value from the XLIFF file.

Each translated language XLIFF file is loaded individually as a separate entity.
**Base XML Must Be Loaded First**

The base Seed Data SD XML file must be successfully loaded before loading any language translation XLIFF files. No new inserts of language rows are performed, only updates to existing rows.

When loading the base seed XML file, the language rows are initially created using the US translation values. Then when loading the language translation files, the rows are updated using the incoming language values. This way, it is not necessary to have translations for every single row. If no translations exist, the fall back is then to use the US language row.

It is not necessary to upload the US language XLIFF file, as US translation data is already saved in the base Seed Data SD XML file.

### 58.3.4 Oracle Fusion Middleware Extensions for Applications Translation Support

For the Seed Data Framework to work with translated data and perform the necessary read and updates to the language translation tables, the application data model must conform to the Oracle Applications Multi-language support guidelines.

See [Understanding Multi-Language Support Features](#) for more information on creating translatable data models.
Use the Database Schema Deployment Framework

This chapter discusses database modeling and database schema deployment in Oracle Fusion Middleware.

When designing an application to interact with the database, you will need to understand the database schema and be able to modify the schema as needed. This chapter contains information regarding database modeling and database schema deployment in Oracle Fusion Middleware. Developers should not use SQL DDL scripts for deployment and source control of database objects, because they tend to be error-prone and do not serve as a single accurate source. Instead, developers should use the JDeveloper offline database schema object files in SXML persistence mode.

Note:
Before SXML migration, these were referred to as XDF (extension) files.

This chapter includes the following sections:

- Introduction to Using the Database Schema Deployment Framework
- Implementing Applications Data Modeling and Deployment JDeveloper Extensions (Data Modeling Extensions)
- Using Schema Separation to Provide Grants

59.1 Introduction to Using the Database Schema Deployment Framework

The Oracle Fusion Schema Deployment framework includes JDeveloper plugins that handle applications-specific metadata, data modeling standards for applications database modeling, and deployment of database schema objects to a target application database. The database schema deployment component can be invoked standalone outside of JDeveloper, such as from the command line, build scripts, or a patching tool.

59.2 Implementing Applications Data Modeling and Deployment JDeveloper Extensions (Data Modeling Extensions)

Oracle uses source-controlled schema metadata files produced from JDeveloper. The Offline Database is a way to persist database object definitions in a JDeveloper project using SXML files, rather than accessing the database directly. It provides an abstract layer that can be used to access a store of database object definitions. Therefore, it is possible to create, edit, delete and manipulate aspects of a database schema offline and access database objects in a database through JDeveloper’s connections.
All schema modeling can be done through JDeveloper. The XDF extension provides developers with a set of tools to do the data physical modeling, such as create, edit, deploy, and import the schema objects used in applications. The extension also provides Application Data Modeling Standard validation, modification, and template object plugins in JDeveloper to help users to follow the Data Model standards. Developers will use XDF extensions for their database modeling development.

Information covered here includes:

- The purpose of the offline database and how to create the database objects in the offline database using JDeveloper.
- The functions provided by the XDF extension how to use the XDF deployment and import tools in JDeveloper.
- The definitions of all User Defined Properties (UDP) that are provided by the XDF extension user property library.
- The UDPS developers should and can define for their schema objects in JDeveloper.

59.2.1 How to Use the Offline Database

JDeveloper provides the tools you need to create and edit database objects, such as tables and constraints, outside the context of a database, using the offline Database model. You can create new tables and views, and generate the information to a database, or you can import database objects from a database schema, make the changes you want, and generate the changes back to the same database schema, to a new database schema, or to a file that you can run against a database at a later date.

59.2.2 How to Create an Offline Database

Follow these directions to create an offline database.

1. In the Application navigator within JDeveloper, locate the project you want to work in.

2. Select File > New to display the New Gallery.

3. From the New Gallery, select Database Tier > Offline Database Objects > Offline Database, as shown in Figure 59-1.
Figure 59-1  Creating a New Offline Database

4. In the Create Offline Database dialog, enter a name for the offline database, as shown in Figure 59-2. For more information at any time, press F1 or click Help from within the Create Offline Database dialog.

Figure 59-2  Naming the New Offline Database

The following types of objects are modeled using offline database objects.

- Table
- Trigger
- View
- Materialized View
JDeveloper offline database objects do not support these objects. However, SXML persistence files for these object types can be imported using the applxdf extension.

59.2.3 How to Deploy an Offline Database in XML Persistence Format

See Deploying in SXML Persistence Format.

59.2.4 How to Validate Application Data Model Standards

A Framework plug-in on the JDeveloper database object editor provides warnings and errors to enforce Data Modeling standards and XDF deployment requirements.

Additional Validations for Schema Object Deployment

To better service the Schema Deployment on Application Data Model, these validations have been added on some User Defined Property (UDP) and other object properties in the plugin.

- Table Owner (UDP)
  
  For a table object, if the User Property Table Owner is not defined, an error will be displayed.

- Short Name (UDP)
  
  All schema objects:
  
  - If the length of the object name is greater than the length standard and if the short name is null or empty, display a warning.
    
    If the short name is not null or empty, check if the length of the short name is greater than the length standard. If it is, display a warning message.

  - If the length of the object name is not greater than the length standard and if the short name is null or empty, automatically set the short name to be the same as the object name.
    
    If the short name is not null or empty, check if the length of the short name is greater than the length standard. If it is, display a warning message.

    The name length standard for a Table is 24; for all others, it is 27.

- Adxml (UDP)
  
  The UDP adxml is set automatically for these schema object types:
The content of the UDP adxml will be determined by the current value of the UDP adxml and the UDP useExistingAdxml.

- AdxmlFK (UDP)
  For table type only, automatically set the UDP adxmlFk according to the values of the UDP adxmlFk and useExistingAdxml.

- AdxmlDeferredIndexes (UDP)
  For table and MaterializedView.type only, automatically set the UDP adxmlDeferredIndexes according to the current value of this UDP and the value of UDP useExistingAdxml.

- Active Constraint or Index’s Columns Checking
  Check the status of a column to which an active constraint or index refers. If its value is obsolete, an error message be displayed and block the work flow.

- Index for Unique Constraint
  Check if an index of unique constraint exists. If not, add one automatically.

- Constraint Deployment Violation Checking
  If a constraint is defined as disabled, but its UDP isLogical is set to N, a warning message will be displayed, because this case will cause a deployment error.

- Dependencies/Risk
  This feature uses JDeveloper’s Offline database APIs and therefore has a dependency on all the offline database JAR files. The risk for this feature is some enforcement of standards may fire wrongly due to potential bugs preventing developers from modeling their objects as needed.

### 59.2.5 Application User Defined Properties

JDeveloper provides a large number of User Defined Properties (UDP). Their mapping with the tables in the dictionary are detailed in:

- User Defined Properties for Tables
- User Defined Properties for Columns
- User Defined Properties for Indexes
- User Defined Properties for Constraints
- User Defined Properties for Views
- User Defined Properties for Sequence
- User Defined Properties for Materialized View
- User Defined Properties for Materialized View Log
- User Defined Properties for Trigger
### 59.2.5.1 User-defined Properties for Tables

Table 59-1 shows the User Defined Properties that are defined for the tables.

<table>
<thead>
<tr>
<th>UDP</th>
<th>Display Name</th>
<th>Values in JDeveloper</th>
<th>Definition in FND_TABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>enableAudit</td>
<td>Enable Audit</td>
<td>Y - Enable Audit. Indicates that audit is enabled on the table. A corresponding shadow table is created. The shadow table is used for maintaining audit on the table. N (default): Audit on the table is not enabled.</td>
<td>ENABLE_AUDIT Not Null</td>
</tr>
<tr>
<td>isFlashbackAllowed</td>
<td>Is Flashback Allowed</td>
<td>Y: Flashback of the table is allowed. That is, you can use Flashback Query to examine the state of a table at a previous time, or use the FLASHBACK TABLE statement to restore an earlier state of a table in the event of human or application error. N (default): Flashback of the table is not allowed.</td>
<td>FLASHBACK_Allowed Not Null</td>
</tr>
<tr>
<td>isLogical</td>
<td>Editioning View</td>
<td>Y (default): The deployment program will create an editioning view for this table. N: The deployment program will not create an editioning view for this table.</td>
<td>LOGICAL VARCHAR2(1)</td>
</tr>
<tr>
<td>runTwice</td>
<td>Run Deployment Twice</td>
<td>Y: Specifies that the table must be deployed twice and the appropriate patching metadata will be stamped in the file. This is typically used when a product team adds or modifies a column as a not-null column to an existing table, and does not want to use a RBMS default value. The column will be populated with an upgrade script having a more complex logic. The column needs to exist in the target database before the upgrade script is run. Also, the upgrade script cannot enforce the not-null constraint since it is against the standards to have DDL in scripts. Setting this UDP to Y will accommodate this.N (default).</td>
<td>N/A</td>
</tr>
<tr>
<td>shortName</td>
<td>Table Short Name</td>
<td>N/A</td>
<td>SHORT_NAME VARCHAR2(30)</td>
</tr>
</tbody>
</table>

#### Notes
- The short name of the table is used by the Zero Downtime programs to uniquely identify the table. The maximum length of this UDP is 24 characters.
- The value of this UDP is defaulted to the table name when the length of the table name is less than 24 characters. When the length of the table name is greater than 24 characters, this UDP is required.
Table 59-1  (Cont.) User Defined Properties for Tables

<table>
<thead>
<tr>
<th>UDP</th>
<th>Display Name</th>
<th>Values in JDeveloper</th>
<th>Definition in FND_TABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectOwner</td>
<td>Table Owner</td>
<td>N/A</td>
<td>APPLICATION_SHHORT_NAME VARCHAR2(10)</td>
</tr>
<tr>
<td>tsClassification</td>
<td>Tablespace Classification</td>
<td>TRANSACTION TABLES (default) REFERENCE INTERFACE SUMMARY ARCHIVE TOOLS MEDIA</td>
<td>N/A</td>
</tr>
<tr>
<td>mlsSupportModel</td>
<td>MLS Support Model</td>
<td>Not MLS (default): The table is not an MLS table. Fully Synched: For Standard MLS (pair of _B and _TL tables) or Single MLS (single _TL) tables. A record will exist in the _TL table for each licensed language in the instance. The _TL table must have LANGUAGE and SOURCE_LANG columns. Partially Synched: For Partially Synchronized MLS tables. These tables have a LANGUAGE column but do not have a SOURCE_LANG column. A record may or may not exist in the table for the licensed languages in the instance. Single Language: For Single Language tables. These tables do not have either a LANGUAGE or a SOURCE_LANG column. The language of the data in the translatable columns in the table is considered to be the language classified as the default or base language of the instance.</td>
<td>MLS_SUPPORT_MODEL VARCHAR2(30)</td>
</tr>
<tr>
<td>status</td>
<td>Status</td>
<td>Active (default): The table is active. Obsolete: The table is obsolete and can be deleted from the database.</td>
<td>STATUS VARCHAR2(30)</td>
</tr>
<tr>
<td>extensionOfTable</td>
<td>Extension of Table</td>
<td>N/A</td>
<td>EXTENSION_OF_TABLE VARCHAR2(30)</td>
</tr>
<tr>
<td>UDP</td>
<td>Display Name</td>
<td>Values in JDeveloper</td>
<td>Definition in FND_TABLES</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>deployTo</td>
<td>Deploy To</td>
<td>The value of this UDP indicates the deployment mode of the table for the Oracle Fusion Disconnected Mobile Framework. This UDP is required.</td>
<td>DEPLOY_TO VARCHAR2(30)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Server DB Only (default): The table is deployed on the server database but not on the mobile database.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>All: The table is deployed both on the server database and on the mobile database.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mobile DB Only: The table is deployed on the mobile database but not on the server database.</td>
<td></td>
</tr>
<tr>
<td>conflictResolution</td>
<td>Conflict Resolution</td>
<td>The value of this UDP indicates how the Oracle Fusion Disconnected Mobile Framework should resolve the conflicts about duplicate rows. This UDP is required.</td>
<td>CONFLICT_RESOLUTION VARCHAR2(30)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Duplicate (default): A new duplicate record is added and the conflict will be handled during the next synchronization. This value is necessary for non-intersection tables. Merge: The records are merged. This value is necessary for intersection tables.</td>
<td></td>
</tr>
<tr>
<td>sharedObject</td>
<td>Shared Object</td>
<td>The value of this UDP indicates whether the table is accessed by external products.</td>
<td>SHARED_OBJECT VARCHAR2(30)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y: The table can be accessed directly by external products, other than the owning product.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>N (default): The table cannot be accessed directly by external products.</td>
<td></td>
</tr>
<tr>
<td>adxml</td>
<td>ADXML</td>
<td>The value of this UDP is patch metadata used by the patching tool.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>axdmlFk</td>
<td>ADXML for Foreign Keys</td>
<td>The value of this UDP is patch metadata used by the foreign key portion of the patching tool.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>adxmlDeferredIndexes</td>
<td>ADXML for Deferred Indexes</td>
<td>The value of this UDP is patch metadata used by the deferred indexes portion of the patching tool.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>useExistingAdxml</td>
<td>Use Existing ADXML</td>
<td>This UDP is mandatory.</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y: Generate ADXML comment using the existing ADXML value from the ADXML UDP</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>N (default): Regenerate the ADXML comment and update the ADXML UDP.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 59-1 (Cont.) User Defined Properties for Tables

<table>
<thead>
<tr>
<th>UDP</th>
<th>Display Name</th>
<th>Values in JDeveloper</th>
<th>Definition in FND_TABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>isSelectAllowed</code></td>
<td>Is Select Allowed</td>
<td>Y (default)N</td>
<td>SELECT_ALLOWED</td>
</tr>
<tr>
<td>This property indicates</td>
<td></td>
<td></td>
<td>VARCHAR2(1) Not Null</td>
</tr>
<tr>
<td>whether the select on</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the table is allowed.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This is mandatory.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>isUpdateAllowed</code></td>
<td>Is Update Allowed</td>
<td>Y (default)N</td>
<td>UPDATE_ALLOWED</td>
</tr>
<tr>
<td>This property indicates</td>
<td></td>
<td></td>
<td>VARCHAR2(1) Not Null</td>
</tr>
<tr>
<td>whether update on the</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>table is allowed.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This property is</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mandatory.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>isInsertAllowed</code></td>
<td>Is Insert Allowed</td>
<td>Y (default)</td>
<td>INSERT_ALLOWED</td>
</tr>
<tr>
<td>This property indicates</td>
<td></td>
<td>N</td>
<td>VARCHAR2(1) Not Null</td>
</tr>
<tr>
<td>whether the insert on</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the table is allowed.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This property is</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mandatory.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>isDeleteAllowed</code></td>
<td>Is Delete Allowed</td>
<td>Y (default)</td>
<td>DELETE_ALLOWED</td>
</tr>
<tr>
<td>This property indicates</td>
<td></td>
<td>N</td>
<td>VARCHAR2(1) Not Null</td>
</tr>
<tr>
<td>whether delete on the</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>table is allowed.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This property is</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mandatory.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>isTruncateAllowed</code></td>
<td>Is Truncate Allowed</td>
<td>Y (default)</td>
<td>TRUNCATE_ALLOWED</td>
</tr>
<tr>
<td>This property indicates</td>
<td></td>
<td>N (default)</td>
<td>VARCHAR2(1) Not Null</td>
</tr>
<tr>
<td>whether truncate on the</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>table is allowed.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This property is</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mandatory.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>maintainPartition</code></td>
<td>Maintain Partition</td>
<td>Y</td>
<td>MAINTAIN_PARTITION</td>
</tr>
<tr>
<td>Indicates whether</td>
<td></td>
<td>N (default)</td>
<td>VARCHAR2(1) Not Null</td>
</tr>
<tr>
<td>partitions can be</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>maintained on the table.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This property is</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mandatory.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>exchangePartition</code></td>
<td>Exchange Partition</td>
<td>Y</td>
<td>EXCHANGE_PARTITION</td>
</tr>
<tr>
<td>Indicates whether it is</td>
<td></td>
<td>N (default)</td>
<td>VARCHAR2(1) Not Null</td>
</tr>
<tr>
<td>possible to exchange</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>partitions on the table.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This property is</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mandatory.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>maintainIndex</code></td>
<td>Maintain Index</td>
<td>Y</td>
<td>MAINTAIN_INDEX</td>
</tr>
<tr>
<td>This property indicates</td>
<td></td>
<td>N (default)</td>
<td>VARCHAR2(1) Not Null</td>
</tr>
<tr>
<td>whether it is possible</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>to maintain indexes on</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the table.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>This property is</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mandatory.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 59.2.5.2 User Defined Properties for Columns

Table 59-2 shows the User Defined Properties that are defined for columns.

<table>
<thead>
<tr>
<th>UDP</th>
<th>Display Name</th>
<th>Values in JDeveloper</th>
<th>Definition in FND_COLUMNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>logging</td>
<td>Supplemental Logging</td>
<td>Y</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>This property indicates if the column needs to run on the supplemental logging during the table creation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>shortName</td>
<td>Column Short Name</td>
<td>N/A</td>
<td>SHORT_NAME VARCHAR2(27)</td>
</tr>
<tr>
<td></td>
<td>The short name of the column is used by the Zero Downtime programs to uniquely identify the column within the table. The value of this UDP is defaulted to the column name when the length of the column name is less than 27 characters. When the length of the column name is greater than 27 characters, this UDP is required.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>translateFlag</td>
<td>Translate</td>
<td>Y: The column is translatable. N (default): The column is not translatable.</td>
<td>TRANSLATE_FLAG VARCHAR2(1) Not Null</td>
</tr>
<tr>
<td></td>
<td>The value of this UDP indicates whether the column is translatable or not. This UDP is required.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>status</td>
<td>Status</td>
<td>Active (default): The column is active. Obsolete: The column is obsolete and can be deleted from the database.</td>
<td>STATUS VARCHAR2(30)</td>
</tr>
<tr>
<td></td>
<td>The value of this UDP indicates the status of the column. This UDP is required.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>customDefaultValue</td>
<td>Custom Default Value</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>denormPath</td>
<td>Denormalization Path</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>The value of this UDP indicates the name of the column that stores the data that should be copied to this column as per the Oracle Fusion Disconnected Mobile Framework.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 59-2 (Cont.) User Defined Properties for Columns

<table>
<thead>
<tr>
<th>UDP</th>
<th>Display Name</th>
<th>Values in JDeveloper</th>
<th>Definition in FND_COLUMNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>routingMode</td>
<td>Routing Mode</td>
<td>Normal (default): The contents of this column must be routed to the destination database. Do Not Route: The contents of this column must not be routed to the destination database.</td>
<td></td>
</tr>
<tr>
<td>histogram</td>
<td>Histogram</td>
<td>Y (default): This column is a candidate for histogram. N: This column is not a candidate for histogram.</td>
<td></td>
</tr>
<tr>
<td>histogramSize</td>
<td>Histogram Size</td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>versionColumn</td>
<td>Disconnected Mobile Version Column Name</td>
<td>N/A</td>
<td>VERSION_COLUMN N VARCHAR2(30 CHAR)</td>
</tr>
<tr>
<td>eligibleToBeSecured</td>
<td>Eligible to be Secured</td>
<td>N (default)</td>
<td>ELIGIBLE_TO_BE_SECURED VARCHAR2(1 CHAR)</td>
</tr>
</tbody>
</table>

### 59.2.5.3 User-defined Properties for Indexes

Table 59-3 shows the User Defined Properties that are defined for the indexes.
### Table 59-3  User Defined Properties for Indexes

<table>
<thead>
<tr>
<th>UDP</th>
<th>Display Name</th>
<th>Values in JDeveloper</th>
<th>Definition in PND_INDEXES</th>
</tr>
</thead>
<tbody>
<tr>
<td>shortName</td>
<td>Index Short Name</td>
<td>N/A</td>
<td>SHORT_NAME VARCHAR2(30)</td>
</tr>
<tr>
<td>deferred</td>
<td>Index Deferred (Y/N)</td>
<td>N (default): The creation of the index will not be deferred. Y: The creation of the index will be deferred.</td>
<td>N/A</td>
</tr>
<tr>
<td>status</td>
<td>Status</td>
<td>Active (default): The index is active. Obsolete: The index is obsolete and can be deleted from the database.</td>
<td>STATUS VARCHAR2(30)</td>
</tr>
<tr>
<td>deployTo</td>
<td>Deploy To</td>
<td>Server DB Only (default): The index is deployed on the server database but not on the mobile database. All: The index is deployed both on the server database and on the mobile database. Mobile DB Only: The index is deployed on the mobile database but not on the server database.</td>
<td>DEPLOY_TO VARCHAR2(30)</td>
</tr>
</tbody>
</table>

### 59.2.5.4 User Defined Properties for Constraints

Table 59-4 shows the User Defined Properties that are defined for the constraints.
### Table 59-4  User Defined Properties for Constraints

<table>
<thead>
<tr>
<th>UDP</th>
<th>Display Name</th>
<th>Values in JDeveloper</th>
<th>Definition in FND_PRIMARY_KEYS or FND_FOREIGN_KEYS</th>
</tr>
</thead>
<tbody>
<tr>
<td>isLogical</td>
<td>Logical Constraint</td>
<td>Y (default) - The deployment program will create the Primary Key/Unique Key/Foreign Key constraint as DISABLE NORELY NOVALIDATE, and create a Check-type constraint as RELY DISABLE NOVALIDATE. N - The deployment program will create the constraint as defined by the developer.</td>
<td>LOGICAL VARCHAR2(1)</td>
</tr>
<tr>
<td>shortName</td>
<td>Constraint Short Name</td>
<td>The short name of the constraint is used by the Zero Down Time patching programs to uniquely identify the constraint. The value of this UDP is defaulted to the constraint name when the length of the constraint name is less than 28 characters. When the length of the constraint name is greater than 28 characters, the developer must enter a value that uniquely identifies the constraint.</td>
<td>SHORT_NAME VARCHAR(30)</td>
</tr>
<tr>
<td>conDefer</td>
<td>Defer Constraint</td>
<td>N (default): The creation of the constraint will not be deferred. Y: The creation of the constraint will be deferred.</td>
<td>N/A</td>
</tr>
<tr>
<td>status</td>
<td>Status</td>
<td>Active (default): The constraint is active. Obsolete: The constraint is obsolete and can be deleted from the database.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

#### 59.2.5.5 User-defined Properties for Views

Table 59-5 shows the User Defined Properties that are defined for the views.
<table>
<thead>
<tr>
<th>UDP</th>
<th>Display Name</th>
<th>Values in JDeveloper</th>
<th>Definition in FND_VIEWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>adxml</td>
<td>ADXML</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>The value of this UDP is patch metadata used by the patching tool.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>isFlashbackAllowed</td>
<td>Is Flashback Allowed</td>
<td>Y: Flashback of the view is allowed. That is, you can use Flashback Query to examine the state of a view at a previous time. N (default): Flashback of the view is not allowed.</td>
<td>FLASHBACK_ALLOWED Not Null</td>
</tr>
<tr>
<td>useExistingAdxml</td>
<td>Use Existing ADXML</td>
<td>Y: Generate ADXML comment using existing ADXML value from ADXML UDP. N (default): Regenerate ADXML comment and update ADXML UDP.</td>
<td>N/A</td>
</tr>
<tr>
<td>objectOwner</td>
<td>View Owner</td>
<td>N/A</td>
<td>APPLICATION_SHORT_NAME</td>
</tr>
<tr>
<td>This UDP stores the Short Name of the application to which the View belongs. This UDP is mandatory.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>status</td>
<td>Status</td>
<td>Active (default): The view is active. Obsolete: The view is obsolete and can be deleted from the database.</td>
<td>STATUS VARCHAR2(30)</td>
</tr>
<tr>
<td>isSelectAllowed</td>
<td>Is Select Allowed</td>
<td>Y (default)N</td>
<td>SELECT_ALLOWED VARCHAR2(1) Not Null</td>
</tr>
<tr>
<td>This property indicates whether the select on the table is allowed. This is mandatory.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>isUpdateAllowed</td>
<td>Is Update Allowed</td>
<td>Y (default)N</td>
<td>UPDATE_ALLOWED VARCHAR2(1) Not Null</td>
</tr>
<tr>
<td>This property indicates whether update on the table is allowed. This property is mandatory.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>isInsertAllowed</td>
<td>Is Insert Allowed</td>
<td>Y (default)N</td>
<td>INSERT_ALLOWED VARCHAR2(1) Not Null</td>
</tr>
<tr>
<td>This property indicates whether the insert on the table is allowed. This property is mandatory.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>isDeleteAllowed</td>
<td>Is Delete Allowed</td>
<td>Y (default)N</td>
<td>DELETE_ALLOWED VARCHAR2(1) Not Null</td>
</tr>
<tr>
<td>This property indicates whether delete on the table is allowed. This property is mandatory.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
59.2.5.6 User-defined Properties for Sequence

Table 59-6 shows the User Defined Properties that are defined for Sequence.

<table>
<thead>
<tr>
<th>UDP</th>
<th>Display Name</th>
<th>Values in JDeveloper</th>
<th>Definition in FND_SEQUENCES</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectOwner</td>
<td>Sequence Owner</td>
<td>N/A</td>
<td>APPLICATION_SHORT_NAME</td>
</tr>
<tr>
<td>status</td>
<td>Status</td>
<td>Active (default): The sequence is active. Obsolete: The sequence is obsolete and can be deleted from the database.</td>
<td>STATUS VARCHAR2(30)</td>
</tr>
<tr>
<td>adxml</td>
<td>ADXML</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>useExistingAdxml</td>
<td>Use Existing ADXML</td>
<td>Y: Generate ADXML comment using existing ADXML value from ADXML UDP. N (default): Regenerate ADXML comment and update ADXML UDP.</td>
<td>N/A</td>
</tr>
<tr>
<td>isSelectAllowed</td>
<td>Is Select Allowed</td>
<td>Y (default)N</td>
<td>SELECT_ALLOWED VARCHAR2(1) Not Null</td>
</tr>
<tr>
<td>resetSequence</td>
<td>Reset Sequence</td>
<td>Y (default)</td>
<td>RESET_SEQUENCE VARCHAR2(1) Not Null</td>
</tr>
</tbody>
</table>

59.2.5.7 User-defined Properties for Materialized View

Table 59-7 shows the User Defined Properties that are defined for the Materialized View.
Table 59-7  User Defined Properties for Materialized View

<table>
<thead>
<tr>
<th>UDP</th>
<th>Display Name</th>
<th>Values in JDeveloper</th>
<th>Definition in FND_MVIEWS</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectOwner</td>
<td>Mview Owner</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Short Name of the application to which this materialized view belongs.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>shortName</td>
<td>Materialized View Short Name</td>
<td>N/A</td>
<td>SHORT_NAME VARCHAR2(30)</td>
</tr>
<tr>
<td>Materialized view short name. Max Length is 24.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>status</td>
<td>Status</td>
<td>Active (default): The materialized view is active. Obsolete: The materialized view is obsolete and can be deleted from the database.</td>
<td></td>
</tr>
<tr>
<td>adxml</td>
<td>ADXML</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>The value of this UDP is patch metadata used by the patching tool.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tsClassification</td>
<td>Tablespace Classification</td>
<td>TRANSACTION_TABLES REFERENCE INTERFACE SUMMARY (default) ARCHIVE TOOLS MEDIA</td>
<td>N/A</td>
</tr>
<tr>
<td>useExistingAdxml</td>
<td>Use Existing ADXML</td>
<td>Y: Generate ADXML comment using existing ADXML value from ADXML UDP. N (default): Regenerate ADXML comment and update ADXML UDP.</td>
<td>N/A</td>
</tr>
<tr>
<td>isSelectAllowed</td>
<td>Is Select Allowed</td>
<td>Y (default)N</td>
<td>SELECT_ALLOWED VARCHAR2(1) Not Null</td>
</tr>
<tr>
<td>This property indicates whether select on the table is allowed. This UDP is mandatory.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>adxmlDeferredIndexes</td>
<td>ADXML for Deferred Indexes</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>The value of this UDP is patch metadata used by the deferred indexes portion of the patching tool.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

59.2.5.8 User-defined Properties for Materialized View Log

Table 59-8 shows the User Defined Properties that are defined for the Materialized View Log.
Table 59-8   User Defined Properties for Materialized View Log

<table>
<thead>
<tr>
<th>UDP</th>
<th>Display Name</th>
<th>Values in JDeveloper</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Status</td>
<td>Active (default): The materialized view log is active. Obsolete: The materialized view log is obsolete and can be deleted from the database.</td>
</tr>
<tr>
<td>objectOwner</td>
<td>MV Log Owner</td>
<td>N/A</td>
</tr>
<tr>
<td>adxml</td>
<td>ADXML</td>
<td></td>
</tr>
<tr>
<td>useExistingAdxml</td>
<td>Y</td>
<td>Y: Generate ADXML comment using existing ADXML value from ADXML UDP. N (default): Regenerate ADXML comment and update ADXML UDP.</td>
</tr>
</tbody>
</table>

59.2.5.9 User-defined Properties for Trigger

Table 59-9 shows the User Defined Properties that are defined for Trigger.

Table 59-9   User Defined Properties for Trigger

<table>
<thead>
<tr>
<th>UDP</th>
<th>Display Name</th>
<th>Values in JDeveloper</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Status</td>
<td>Active (default): The trigger is active. Obsolete: The trigger is obsolete and can be deleted from the database.</td>
</tr>
<tr>
<td>objectOwner</td>
<td>Trigger Owner</td>
<td>N/A</td>
</tr>
<tr>
<td>adxml</td>
<td>ADXML</td>
<td></td>
</tr>
<tr>
<td>useExistingAdxml</td>
<td>Use Existing ADXML</td>
<td>Y: Generate ADXML comment using existing ADXML value from ADXML UDP. N (default): Regenerate ADXML comment and update ADXML UDP.</td>
</tr>
</tbody>
</table>

59.2.6 How to Create an Offline Database Object

To create new offline database objects from within JDeveloper, in the Application Navigator:

1. Right-click the offline Database or schema.
2. Select New Database Object.
3. Select any Offline database object definition that you wish to create.
You also can create an offline database object definition by importing an existing definition from an online database schema.

### 59.2.7 How to Edit an Offline Database Object

To edit an offline database object:

1. In the Application Navigator, expand the workspace, project, and schema.

2. Right-click the offline database object that you wish to edit and choose **Properties**. Or double-click the offline database object.

   The Edit offline database object dialog opens. For more information at any time, press **F1** or click Help from within the Edit dialog.

3. In the Edit dialog, select an information category on the left and change the values in the panel on the right. Any items that are grayed out cannot be selected or changed.

### 59.2.8 How to Import an Offline Database Object

Database objects from a database schema can be imported to an offline database project in JDeveloper. JDeveloper extensions will also handle the additional Oracle Fusion metadata, if available in the target database. The additional metadata will be copied to an offline database object as user-defined properties. For objects not supported by JDeveloper, such as Policy and Advanced queue tables, import of object definitions from the database will be provided. Implementation of unsupported object import process APIs depends on functionality provided by the Metadata team (dbms_metadata).

**Using the Import Offline Database Object Wizard**

Object definitions can be generated to the Offline Database by right-clicking an Offline Database Source and selecting the **Reverse Engineer Fusion Applications Objects** option, extended to invoke the relevant import API, as shown in Figure 59-3.

*Figure 59-3    Starting the Import Database Object Wizard*
1. The target database connection name can be selected from the list of all defined database connections, or a new connection can be created in the Specify Source dialog, shown in Figure 59-4.

Figure 59-4   Specifying the Source

2. Select the Project and Offline database to which the objects from the target database need to be imported, as shown in Figure 59-5.

Figure 59-5   Specifying the Target

Filters can be applied to select the objects that are displayed as available for import. When there are a large number of objects in the schema, you should apply filters. In the Object Picker, shown in Figure 59-6, you can:
• Enter characters in the Name Filter to filter the list of available objects by name. The Name Filter is case sensitive.

• When there are a large number of objects, you can turn off Auto-Query and click **Query** when you have entered the filter you want to use.

• Select the object types you want to view.

**Figure 59-6  Picking an Object**

3. Click **Next** to display the summary information.

4. Click **Finish** to import the selected objects to the specified offline database.

### 59.2.9 How to Deploy the Offline Database Objects

When an offline database object is created, it can be deployed to a target database using the deployment extension provided in JDeveloper.

#### 59.2.9.1 Deploying in SXML Persistence Format

As part of the Oracle Middleware Extensions for Applications (Applications Core) labels, the `applxdf` extensions for the JDeveloper offline database include a deployment program that operates on JDeveloper offline database objects in SXML format. It checks for and compares the object definitions in SXML format with the object definitions in the target database, and then executes the necessary create/alter DDL to deploy the objects. This deployment program is available in two forms:

• Deployment wizard extension that can be invoked from within JDeveloper.

• Standalone deployment program that can be invoked from the command line.

##### 59.2.9.1.1 How to Use the Database Object Deployment Wizard in JDeveloper

To start the deployment wizard in JDeveloper, you need to choose **APPS: Deploy DB Object** from the context menu on the offline object definition in the Application Navigator.
This can be used to deploy an offline database to a target online database. These options are available for deployment:

- Deploying a single database object file. Only one item is selected, as shown in Figure 59-7.

**Figure 59-7  Deploying a Single Database Object File**

- Deploying multiple database object files (Bulk Deployment). Several items are selected, as shown in Figure 59-8.
• Deploying offline schema object (and thereby deploying the entire designed data model), as shown in Figure 59-9.
The Generate Fusion Applications Objects wizard will prompt for the following information:

- **Database Connection**: The target database connection name can be selected from the list of all defined database connections, or a new connection can be created, as shown in Figure 59-10.
• Deployment Parameters. Figure 59-11 shows how the dialog appears for single database deployment parameters.

**Figure 59-11 Single Database Object Deployment Parameters**

Figure 59-12 shows how the Deployment Parameters dialog appears for multiple database deployment parameters.
Owner User: Oracle schema name in which the object exists or should be created.

Log File Path: Specify a logfile name if it has to be written to a log file. By default, the log will be displayed in the JDeveloper log window.

For Single Database Object deployment, this is optional. For bulk Database Object and schema deployment, it is mandatory to provide a directory for saving log files.

Log File Format: The format of the log file. Permitted values are text (the default) or xml.

Debug Level: The Debug level controls the level of detail to be captured in the log. Debug Level=3 will show the most information. Permitted values are 0 (the default), 1, 2 or 3.

Db Object Mode: A single database object can be deployed independently in table and in tablefk mode.

In case of bulk and schema deployment, the database objects first will be deployed in table mode and then tablefk mode, by default.

This is standard for bulk and schema deployment; therefore, the Db Object Mode is not displayed on the wizard.

Stand Alone: If this option is selected, the XDF comparison utility will execute in a standalone mode. This mode does not have any applications dependencies and it creates database objects without applications standards for physical attributes such as TABLESPACE/STORAGE; the database defaults are used.
- **Change Database**: This option indicates whether the deployment should just report or execute the necessary Alter DDLs, based on comparison of the offline Database object definition against target database. If unchecked, the deployment will report on the differences but will not actually apply the changes to the database.

- **Force Mode**: If this option is selected, any additional column, index or constraints that are present in a target database, but not in the current object file, will be dropped.

### 59.2.9.1.2 How to Use the Database Object Deployment Command Line Interface

Use this command and parameters shown in Example 59-1 to deploy a database object from the command line.

**Mandatory Arguments**

- **owner_un**: Oracle schema name in which the object exists or should be created.
- **apps_un**: Oracle schema name of the current APPS schema.

Note that in the consolidated fusion schema model, owner_un will be the same as apps_un. These parameters are maintained for cases where they could be different.

- **jdbc_protocol**: The JDBC protocol (thin or oci8).
- **jdbc_db_addr**: JDBC tns information, either formatted as a Net8 connect string enclosed in double quotes, or as hostname:port:oracle_sid.
- **xdf_mode**: The object type information - table, qtable, mview, mviewlog, sequence, type, trigger, view, policy, bootstrap.
- **xdf_file_name**: The XDF file name, which contains the object definition. It is not mandatory if the xdf_mode is bootstrap.

**Password Arguments**

The command line deployment tool will prompt to get the database password from the user in an interactive mode. The password cannot be a parameter, because it is against Security guidelines. If you have a script in which you are invoking schema deployment, you can pipe the password in the script, such as:

```
$JDEV_JAVA_HOME/bin/java oracle.apps.fnd.applxdf.comp.XdfSchemaDeploy owner_un=
$FUSION_SCH apps_un=$FUSION_SCH jdbc_protocol=thin jdbc_db_addr=$JDBC_ADDR
changedb=y logfileformat=text xdf_file_name=$xdfFile xdf_mode=$xdf_mode logfile=
$logfile <<! $FUSION_PASS
```

**Optional Parameters**

- **xdf_xsl_dir**: The XSL directory, which contains all the XSL files required for XSLT transformation. This parameter is optional and is automatically determined in most cases if the JAR file format of deploying Java class files is being used. This parameter is maintained for flexibility, in case the format for deploying Java files becomes similar to what existed in previous versions.

- **standalone**: This option is used to execute the XDF comparison utility in a standalone mode. Permitted values are y, Y, n or N. The default value is n. Standalone=y does not have any applications dependency. This mode creates database objects without applications standards for physical attributes such as
TABLESPACE/STORAGE and uses the database defaults. It also does not update applications metadata.

- `changeDb`: The default is "y." If changedb is specified as "n," the SQL statements generated by the XDF comparison utility are not executed but are displayed on the standard output or a log file.

- `logfileformat`: The format of the log file. Permitted values are text or xml. If logfileformat is not set, the default is text.

- `logfile`: The output of the comparison utility is written to standard out. Specify a logfile name if it has to be written to a log file. If logfile is not set, the outputs will be displayed on the screen.

- `debuglevel`: Debug levels determine how much information is to be shown in the log. Debuglevel=3 will show the most information. Permitted values are 0, 1, 2 or 3. The default value is 0.

- `from_jdev`: This parameter is set to "y" when the XDF comparison utility is called from JDeveloper. The default value is "n."

- `force_mode`: The force deployment mode introduces an additional input parameter that when specified will drop any additional column, index or constraints that are present in a target database. The applications metadata stored for the object is also updated to be in sync with the new definition.

- `index_category`: Values are small, large, and both. If the table is partitioned, the index is always created. If the table is not partitioned, there is no index creation if one of these conditions is true:
  - index_category=small and unused dbms block size greater than parallel_index_threshold
  - index_category=large and unused dbms block size less than parallel_index_threshold

- `parallel_index_threshold`: Parallel index threshold, default is 0.

- `no_error`: This option is used when you add a not-null column to an existing table with data, or change a null column to a not-null column. XdfSchemaDeploy results in FAILURE without this option. XdfSchemaDeploy results in WARNING if you have no_error=y. Default value is "n."

- `idxnolog`: Pass idxnolog=y to add a NOLOGGING clause in the Index creation to improve the performance of creation. The default value is "n."

**Example 59-1 Sample Database Object Deployment Using the CLI**

```java
java oracle.apps.fnd.applxdf.comp.XdfSchemaDeploy <owner_un={schemaId}>
<apps_un={appId}> <jdbc_protocol={jdbc driver type}> <jdbc_db_addr={jdbc tns info}>
<xdf_file_name={xdf file name}> <xdf_mode={xdf mode}> [xdf_xsl_dir={xsl file directory}] [standalone={y|n}] [changedb={y|n}] [logfileformat={text|xml}]
[logfile={log file path and name}] [debuglevel={0|1|2|3}] [from_jdev={y|n}]*
```

### 59.2.9.2 Setting the CLASSPATH Variable

The XdfSchemaDeploy tool requires JDK 1.6, the standard Oracle JDBC driver, the XML parser, and the applxdf JAR file.

Set the CLASSPATH environment variable to contain these JAR files:
Example 59-2 shows the set of commands to set the CLASSPATH.

The JDeveloper installation directory could potentially change with newer versions of JDeveloper being available. Check the JDeveloper installation directory to make sure that it exists. You could also use the XML parser and JDBC driver that comes with the database. Note that so far XDF has been tested with the same JAR files with which it has been compiled. It should not be a problem setting the CLASSPATH with a higher version of these JAR files and testing them. If you encounter any issues, try using 11g JDBC and xmlparsers.

**Example 59-2  Example of Setting CLASSPATH**

```
setenv CLASSPATH $ADE_VIEW_ROOT/fmwtools/BUILD_HOME/oracle_common/modules/oracle.nlsrtl_11.1.0/orai18n.jar:
$ADE_VIEW_ROOT/fmwtools/BUILD_HOME/oracle_common/modules/oracle.xdk_11.1.0/xmlparserv2.jar:
$ADE_VIEW_ROOT/fmwtools/BUILD_HOME/wls/server_10.3/server/ext/jdbc/oracle/11g/ojdbc6.jar:
$ADE_VIEW_ROOT/fmwtools/BUILD_HOME/oracle_common/modules/oracle.odl_11.1.1/ojdl.jar:
$ADE_VIEW_ROOT/fmwtools/BUILD_HOME/oracle_common/modules/oracle.odl_11.1.1/ojdl2.jar:
$MW_HOME/jdeveloper/jdev/extensions/oracle.apps.fnd.applxdf.jar
```

### 59.2.9.3 Using Bootstrap Mode

Bootstrap mode for XDF Schema deployment is available using the parameter xdf_mode=bootstrap. XDF currently depends on several database components, such as pl/sql and tables, for it to work completely in all modes. The bootstrap mode can be used to make sure that the XDF database dependencies are set up correctly and to avoid any manual steps to get XDF working on a particular database.

In bootstrap mode, the mandatory parameter xdf_file_name becomes optional and only the remaining mandatory parameters are applicable.

To run XDF in bootstrap mode, use the command shown in **Example 59-3**.

**Example 59-3  Sample of Running XDF in Bootstrap Mode**

```
$JDEV_JAVA_HOME/bin/java oracle.apps.fnd.applxdf.comp.XdfSchemaDeploy
owner_un=fusion apps_un=fusion jdbc_protocol=thin jdbc_db_addr={jdbc tns info}
xdf_file_name="dummy" xdf_mode=bootstrap runtime_schema=<runtime schema name>
```

### 59.2.9.4 Deployment FAQ

Examining these frequently-asked questions about deployment will help you prevent and fix problems.

**To add a not-null column to an existing table with data, or change a null column to a not-null column:**

There are two options to add a not-null column to an existing table with data, or change a null column to a not-null column.
• Option 1
  – Adding a not null column to an existing table with data
    Product teams can specify an RDBMS default value for the column which will be used by the databases to successfully alter the table to add the not-null column.
  – Modifying a null column to not-null in a table with data
    Product teams can specify an RDBMS default value for the column. This default value will be used by schema deployment utility to update the existing null rows with that value before changing the column to not null.

• Option 2
  If a product team does not want to use a RDBMS default value, it can add or modify a column as a not-null column to an existing table by using a script having a more complex logic. The column needs to exist in the target database before the script is run. The script cannot enforce the not-null constraint because it is against the standards to have DDL in scripts.

  To use a script to populate the column, the UDP named runTwice must be set to Yes. This UDP will be used by XDF to ensure that the required patch metadata is present in XDF to run it twice in a patch. In the first run, the script will not error out if it is not able to enforce the not-null constraint, but it will error out in the second run if it still is not able to enforce the not-null constraint.

  If the user does not set this UDP, the default behavior of the deployment utility is to error out if it is not able to enforce the not-null constraint while adding or modifying the column.

To remove a table, column or view:
See How to Use fnd_cleanup_pkg and fnd_drop_obsolete_objects.

To rename a table, column or view:
Deployment does not support renaming a table, column or view. The workaround that can be used is to make the object obsolete and introduce a new renamed object. The development team must separately handle data migration and update information in the Automatic Diagnostic Repository (ADR), if required.

To implement a non-additive change to the data type of a column, such as varchar2 to number:
Developers can create a script that runs before deployment of the object to rename or drop the column, based on whether data must be preserved or migrated. When the deployment successfully adds the column with the correct data type, another script may be needed to make sure that data is migrated and that the renamed column is dropped. This is applicable only if the initial script renames the column.

To add or change the unique constraints or indexes on a populated table:
If the populated table does not meet the criteria for creating unique constraint or unique index, to remove the invalid data create a script to clean the table before deploying a unique index or constraint.

59.2.9.5 Cleaning Database Objects
To maintain efficiency, database objects should be cleaned of no-longer-used data. As part of that process, it is important to keep the FND data dictionary synchronized with
existing objects (it can be table, sequence or view) in the database. The XDF team provides packages to make this process easier.

**59.2.9.5.1 Making a Database Object Obsolete**

Use this information to correctly make a database object obsolete. It is applicable after the release of the initial version of the product to customers.

Modeling database schema for new releases or upgrades to new releases may involve making certain database objects or certain attributes of the database object, such as Columns, obsolete. Doing this may make a significant effect to existing customizations or extensions currently implemented on the system. Making obsolete database objects that contain data, such as Tables, requires particularly close analysis for potential effects. Development teams should take the necessary steps to review and understand the implications of such updates in these areas.

Making obsolete certain database objects or certain attributes of a database object may require those objects to be dropped as part of clean up. Considering any existing customization or extensions to such objects, the act of making obsolete and dropping the object should be kept separate. Dropping the obsolete database objects or columns should be an optional step that is invoked at the demand of customers. The Patching (AD) utilities will provide such an option as a post-patching step.

Follow these steps to make obsolete a database object or attribute.

1. Set Status User Defined Property for the specific database objects or attributes to Obsolete. This can be done using User Defined Properties for Applications specific metadata that is part of the offline database model.

2. Status User Defined Property will be captured in the Applications/XDF data dictionary as part of deploying XDF to the database.

These steps ensure that:

- If the table does not already exist in the database, the Applications schema deployment utilities (XDF) will create the table without columns marked as obsolete.

- If the column does not already exist in the table, the Applications schema deployment utilities (XDF) will not add the column to the table.

- If the column already exists in the table, the Applications schema deployment utilities (XDF) will remove any NOT NULL, PK/FK constraints on the column.

- Any indexes that comprise only the obsolete columns could be dropped. In other cases, the development team owning the obsolete objects will be expected to update the definition of any affected indexes.

**59.2.9.5.2 How to Use the Force Mode Option in Schema Deployment**

During the initial development of the product before release, product teams may not want to mark the object as obsolete and may prefer directly dropping the object; that is, removing the definition from the offline database file. To support this, the force_mode=y parameter can be passed to the schema deployment tool. The additional input parameter which, when specified, will drop any additional column, index or constraints that are present in a target database and not present in the offline object file definition. The XDF dictionary metadata stored for the object is also updated to be synchronized with the new definition. Note that this option, in certain cases, will not change the definition of the table to exactly match the definition in the file. For example, if the database does not allow some changes, such as changing of certain
column datatype, or changing an unpartitioned table to a partitioned table, force mode will not override the database.

Force mode only handles the removal of column, index and constraints, and synchronizing the corresponding definition in the XDF dictionary. If the primary object, such as a table, sequence or view, is dropped, the fnd_cleanup_pkg must be used to synchronize the XDF dictionary.

59.2.9.5.3 How to Use fnd_cleanup_pkg and fnd_dropObsolete_objects

Use fnd_cleanup_pkg and fnd_dropObsolete_objects to clean a database.

Using fnd_cleanup_pkg

Procedure fndcleanup(name): Remove table, sequence, or view with name that is in fnd_tables, fnd_views or fnd_sequences tables, but not in the database. All the required XDF dictionary tables will be updated when fnd_tables, fnd_views, or fnd_sequences is updated.

name: Optional, but using fnd_cleanup_pkg.fndcleanup without the name parameter may cause some issues. You should use fndcleanup(object name) for a clean and safe removal. This can be a table name, a view name or a sequence name. If a name is provided, the procedure will clean only the named object and related components. If it is not defined, the fndcleanup procedure removes all table, view, or sequences that are in fnd_tables, fnd_views, or fnd_sequence tables, but that do not exist in the database.

It also removes the entries that refer to that name object in all related database schema deployment (xdf) dictionary tables, including fnd_columns, fnd_indexes, fnd_primary_keys, fnd_primary_key_columns, fnd_foreign_keys, fnd_foreign_key_columns, and fnd_histogram_cols.

Procedure clean_fndcons(tname): Remove the tname table’s Primary Key (PK), Unique Keys (UKs), and Foreign Keys (FKs) that are not in the database from the fnd constraint dictionary tables. Although the tname parameter is optional, you should use it. If tname is not specified, then all tables’ PK, UKs, and FKs that are not in the database will be removed from the fnd constraint tables.

Procedure TableFndCleanUp(tname, deleteType, delname): Remove the specified deleteType with the specified delname on the specified tname table from related fnd tables. Although the delname parameter is optional, you should include it. If it is not specified, all table properties on that deleteType will be removed from the specified table. The deleteType includes column, index, pkuk, and fk.

Examples

• Remove table1 from the fnd_tables table if table1 is not in the database. All required XDF dictionary tables will be updated when fnd_tables is updated.
  
  execute fnd_cleanup_pkg.fndcleanup('table1')

• Remove view1 from the fnd_views table if view1 is not in the database.
  
  execute fnd_cleanup_pkg.fndcleanup('view1')

• Remove all table names LIKE HZ% in fnd_tables or fnd_views tables that do not exist in the database.
  
  execute fnd_cleanup_pkg.fndcleanup('HZ%')

• Remove table XF1’s PK, UKs, and FKs that are not in the database, from the fnd constraint dictionary tables.
exec fnd_cleanup_pkg.clean_fndcons('XF1');

- Remove a Foreign Key named XF2_T1_FK on table XF2 from the fnd constraint dictionary tables. XF2_T1_FK may or may not be in the database.
  
  exec fnd_cleanup_pkg.tablefndcleanup('XF2', 'fk', 'XF2_T1_FK');

**fnd_drop_obsolete_objects**

**Procedure drop_object(objectname):** Drop obsolete objectname from the database. This procedure is used to delete obsolete views, tables and columns that are marked as Obsolete in the table.

**Examples**

- Drop Table table1 from the database if it is marked as obsolete. If table1 is not obsolete, drop any columns in table1 that are obsolete.
  execute fnd_drop_obsolete_objects.drop_object(table1)

- Drop View view1 from the database if it is marked as obsolete.
  execute fnd_drop_obsolete_objects.drop_object(view1)

- Verify all tables and views with name like 'HZ_\%' for dropping tables and views, or drop columns if the tables are not obsolete.
  execute fnd_drop_obsolete_objects.drop_object('HZ_\%')

**59.2.9.5.4 Frequently Asked Questions**

Use this information when dropping an object in the database.

- How do I make an object obsolete?
  1. Select the object in JDeveloper.
  2. Right-click and select **Properties > User Properties**.
  3. Change the Status to **Obsolete**.

- What happens when you deploy an obsolete object or an object that has obsolete columns or indexes?
  Only the FND dictionary is updated. Objects in the database are not dropped.

- How do I remove an object from the database and keep the FND data dictionary synchronized?
  There are three ways this can be done. SQL scripts can be used to achieve the same outcome.
    - If the object is not a primary object and is a column, index or constraint that is being dropped:
      - Use JDeveloper to remove the definition of these secondary objects from the offline database file definition.
      - Use the **force_mode** optional parameter to deploy the object to the target database.
    - If the object is a primary object, such as a table, sequence or view:
* Drop the object obj from sqlplus.
* Execute this command from sqlplus:
  
  ```sql
  execute fnd_cleanup_pkg.fndcleanup(' obj')
  ```

- Change the object obj status UDP to Obsolete from JDeveloper. Deploy the object to the database either by command line (XdfSchemaDeploy) or use Generate Fusion Applications Objects from JDeveloper.

* Execute this command from sqlplus:
  
  ```sql
  execute fnd_dropObsolete_objects.drop_object(' obj')
  ```

* Execute this command from sqlplus:
  
  ```sql
  execute fnd_cleanup_pkg.fndcleanup(' obj')
  ```

- How do I clean up the FND data dictionary if many objects are no longer in the database?
  
  Execute this command from sqlplus or use a SQL script.
  
  ```sql
  execute fnd_cleanup_pkg.fndcleanup
  ```

- How do I clean up the FND data dictionary if I had deleted columns and indexes from table, but did not cleanup from the FND data dictionary.
  
  Execute this command from sqlplus or use a SQL script.
  
  ```sql
  execute fnd_cleanup_pkg.fndcleanup
  ```

- How do I remove obsolete columns/indexes in objects from FND data?
  
  Execute this command from sqlplus.
  
  ```sql
  execute fnd_cleanup_pkg.fndcleanup
  ```

- How do I remove all table/view/sequence name LIKE HZ% in fnd_tables/fnd_views/fnd_sequences tables that do not exist in the database.
  
  Execute this command from sqlplus.
  
  ```sql
  execute fnd_cleanup_pkg.fndcleanup('HZ%')
  ```

- How do I drop table/view/sequence name xyz in fnd_tables/fnd_views/fnd_sequences tables from the database if it is marked as obsolete?
  
  Execute this command from sqlplus.
  
  ```sql
  execute fnd_dropObsolete_objects.drop_object('xyz')
  ```

- How do I drop table/view/sequence name LIKE HZ% in fnd_tables/fnd_views/fnd_sequences tables from the database if it is marked as obsolete?
  
  Execute this command from sqlplus.
  
  ```sql
  execute fnd_dropObsolete_objects.drop_object('HZ%')
  ```

### 59.3 Using Schema Separation to Provide Grants

The application runtime schema could be different from the database object owning schema for security reasons. To support this model as part of schema deployment, there is a mechanism to granularly provide grants on various database objects. These are granted to a set of fixed roles which are eventually available to runtime schema.
Privilege will be granted on the database object to the role based on privilege User defined properties defined for the object. Table 59-10, Table 59-11, Table 59-12, and Table 59-13 present the user defined properties that will be defined for each object type.

**Table 59-10  ** Table Object Type Properties

<table>
<thead>
<tr>
<th>UDP Name</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert Allowed</td>
<td>Grant Insert Privilege on the table to the required role</td>
<td>Y/N Default Y</td>
</tr>
<tr>
<td>Update Allowed</td>
<td>Grant Update Privilege on the table to the required role</td>
<td>Y/N Default Y</td>
</tr>
<tr>
<td>Delete Allowed</td>
<td>Grant Delete Privilege on the table to the required role</td>
<td>Y/N Default Y</td>
</tr>
<tr>
<td>Select Allowed</td>
<td>Grant Select Privilege on the table to the required role</td>
<td>Y/N Default Y</td>
</tr>
<tr>
<td>Truncate Allowed</td>
<td>The value of this UDP indicates whether a TRUNCATE statement is allowed on the table</td>
<td>Y/N Default Y</td>
</tr>
<tr>
<td>Maintain Partition</td>
<td>The value of this UDP indicates whether partitions can be maintained on the table. The ADM_DDL program when handling requests for dynamic DDL operations uses this information.</td>
<td>Y/N Default Y</td>
</tr>
<tr>
<td>Exchange Partitions</td>
<td>The value of this UDP indicates whether it is possible to exchange partitions on the table. The ADM_DDL program when handling requests for dynamic DDL operations uses this information.</td>
<td>Y/N Default Y</td>
</tr>
<tr>
<td>Maintain Index</td>
<td>The value of this UDP indicates whether it is possible to maintain indexes on the table. The ADM_DDL program when handling requests for dynamic DDL operations uses this information.</td>
<td>Y/N Default Y</td>
</tr>
</tbody>
</table>

**Table 59-11  ** View Object Type Properties

<table>
<thead>
<tr>
<th>UDP Name</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert Allowed</td>
<td>Grant Insert Privilege on the view to the required role</td>
<td>Y/N Default Y</td>
</tr>
</tbody>
</table>
### Table 59-11  (Cont.) View Object Type Properties

<table>
<thead>
<tr>
<th>UDP Name</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update Allowed</td>
<td>Grant Update Privilege on the view to the required role</td>
<td>Y/N Default Y</td>
</tr>
<tr>
<td>Delete Allowed</td>
<td>Grant Delete Privilege on the view to the required role</td>
<td>Y/N Default Y</td>
</tr>
<tr>
<td>Select Allowed</td>
<td>Grant Select Privilege on the view to the required role</td>
<td>Y/N Default Y</td>
</tr>
</tbody>
</table>

### Table 59-12  Sequence Object Type Properties

<table>
<thead>
<tr>
<th>UDP Name</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Allowed</td>
<td>Grant Select Privilege on the sequence to the required role</td>
<td>Y/N Default Y</td>
</tr>
<tr>
<td>Reset Sequence</td>
<td>The value of this UDP indicates if the sequence can be reset to a specific value. The ADM_DDL program when handling requests for dynamic DDL operations uses this information.</td>
<td>Y/N Default Y</td>
</tr>
</tbody>
</table>

### Table 59-13  Materialized Views Object Type Properties

<table>
<thead>
<tr>
<th>UDP Name</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Allowed</td>
<td>Grant Select Privilege on the materialized view to the required role</td>
<td>Y/N Default Y</td>
</tr>
</tbody>
</table>
This chapter provides guidelines for you to write high-performing, highly scalable, and reliable applications on Oracle Fusion Middleware.

This chapter includes the following sections:

- Introduction to Improving the Performance of Applications
- ADF Business Components Guidelines
- ADF ViewController Layer Guidelines
- SOA Guidelines for Human Workflow and Approval Management Extensions
- Oracle Fusion Middleware Extensions for Applications Guidelines
- General Java Guidelines
- Caching Data
- Profiling and Tracing Oracle Fusion Applications
- Setting up a Debug Breakpoint

60.1 Introduction to Improving the Performance of Applications

The outcome of performance assessments of several prototypical Oracle Fusion Applications as well as various tests conducted by the Oracle Fusion Middleware performance team are captured in this chapter. It includes best practices for coding and tuning the Oracle Application Development Framework (Oracle ADF) Business Components-based applications with performance, scalability, and reliability (PSR) in mind. Other topics discussed include performance improvement guidelines for ADF ViewController layers and Oracle Fusion Middleware Extensions for applications.

This chapter assumes you are familiar with the concepts described in the Oracle Fusion Middleware Tuning Performance guide.

60.2 ADF Business Components Guidelines

To maximize performance while working with ADF Business Components, such as entity objects, view objects, application modules, and services, consider the best practices described in this section. For more information about tuning Oracle ADF, see the “Oracle Application Development Framework Performance Tuning” chapter in the Tuning Performance.

60.2.1 Work with Entity Objects

When working with entity objects, consider the suggestions for improving performance in this section. For more information see the "What You May Need to
Know About Optimizing View Object Runtime Performance” in the Developing Fusion Web Applications with Oracle Application Development Framework.

60.2.1.1 Enable Batch Updates for your Entity Objects

You can enable batch updates of your entity objects by selecting the **Use Update Batching** property on the **Entity Object Editor - Tuning** section as shown in the following figure. You can also set the **Threshold** property to 1, which is important for _TL multi language entities.

**Figure 60-1    Entity Object Editor — Tuning**

When enabled, ADF Business Components combines multiple Data Manipulation Language (DML) operations and executes them in a single round trip. Modified rows are grouped into batches.

Enable batch updates, except in the following three cases:

- You override the DML (PL/SQL entity objects are in this category).
- You have one or more streaming attributes, such as character large object (CLOB) or binary large object (BLOB).
- One or more attributes are marked as retrieve-on-insert or retrieve on-update.

For more information about batch processing, see the "Batch Processing" section in the Tuning Performance.

60.2.1.2 Avoid Children Entity Objects in Composite Entity Associations Setting the Foreign Key Attribute Values of the Parent

Children entity objects can expect that their parent primary key attribute values are passed through the **attributeList** parameter in `create(attributeList)` and ADF Business Components calls `super.create(attributeList)` to populate these foreign key attribute values. Repopulating the foreign key attribute values in the children entity object unnecessarily decreases performance.

60.2.1.3 Avoid Using List Validator Against Large Lists

When you use list validator, it scans the values in a linear fashion. Therefore, limit the list to not more than 20 to 30 values for frequently used list validators. Instead of using the list validator, you can use either an expression validator or a method validator. If this is a foreign key, then you can use a key exist validator.

60.2.1.4 Avoid Repeated Calls to the same Association Accessor

There is some cost to getting the parent or children via the association accessor. For example, if you are calling the same association accessor on the same entity object in a loop, then move the call to outside the loop.
60.2.1.5 Close Unused RowSets

There are various places in ADF Business Components that loop through all rowsets of a view object. Call RowSet.closeRowSet on any rowsets that you no longer need. The typical case where you would have opened a rowset is when you get the "many" end of an association, for example, when retrieving Emp from Dept.

By default, the rowset is cleared when a garbage collection occurs. However, if you can close the rowset as soon as you finish using it, it improves performance and reduces the amount of work done during garbage collection. To close a rowset, call RowSet.closeRowSet. You should close it only if you know you no longer need it, and would not make calls such as previous(). If you are getting a row iterator from an association accessor, you can cast it to a RowSet and call closeRowSet on it.

Caution:

If you use the Retain Association Accessor RowSet option, then you should not call closeRowSet.

The same principle applies for view accessor rowsets. If your view accessor returns more than one row, then every time you start fetching rows from the view accessor with a different bind value, a new row set is created, along with a new underlying database cursor to execute the query. If you do not need all the rows, then you can configure the MaxFetchSize setting on the view accessor to include only the number of rows you need. Then Oracle ADF closes the database cursor immediately upon fetching all the rows up to MaxFetchSize. Alternatively, for cases where you expect to execute the view accessor several times with different bind values, in a single request, consider calling closeRowSet on the view accessor rowset explicitly.

60.2.1.6 Use "Retain Association Accessor RowSet" when Appropriate

By default, the entity object creates a new RowSet object each time you retrieve an entity association accessor rowset, to allow you to work with the rows. However, creating a new RowSet object does not imply re-executing the query to produce the results each time, since only a new instance of a RowSet object is created, with its default iterator reset to the "slot" before the first row. There is some overhead to creating all these new rowsets even though the ones not in use are cleared on Java Virtual Machine (JVM) garbage collections. You may also see additional query executions due to the rowsets, and resulting underlying query collections, not being retained.

For high-traffic entity objects, such as those used for bulk loading, where the same association accessor is called many times, consider using the Retain Association Accessor RowSet option to improve performance. Typically, an association accessor would be used multiple times if:

- Your entity object is Multi-Language Support (MLS)-enabled and has more than one translated attribute.
- You have defaulting logic or multiple validators that need to access the same association attribute.

Using the Retain Association Accessor RowSet option may adversely affect memory usage since it postpones when the retained rowset becomes garbage collectible. Before you enable this option, as shown in the following figure, profile your flow to make sure you would indeed get a noticeable benefit.
When sorting by exclusive CPU, if you see the top CPU consumers are related to code that loops through the rowsets, then you would likely get a benefit by using this option. An example where you may want to consider using the **Retain Association Accessor RowSet** option is if you profile your code and see that `oracle.jbo.server.ViewObjectImpl.addRowSet` is using a lot of CPU, and most of the CPU is in a call stack that includes `AssociationDefImpl.get`. The following figure illustrates an example profiler output showing `addRowSet` being expensive.

Before you decide to retain association accessor, try the guideline in **Close Unused RowSets**.

If you decide to use the **Retain Association Accessor RowSet** option, ensure you are aware of the potential behavior changes.

For information related to the potential behavior changes as the result of using the **Retain Association Accessor RowSet** option, see the **Advanced Entity Association Techniques** section in the "Advanced Entity Object Techniques" chapter of the Developing Fusion Web Applications with Oracle Application Development Framework.

### 60.2.1.7 Mark the Change Indicator Column

If your table has an `OBJECT_VERSION_NUMBER` column, make sure you check the **Change Indicator** attribute property. Columns marked as **Change Indicator** are automatically in any view object that includes that particular entity object.

### 60.2.2 Work with View Objects

When working with view objects, consider the suggestions in this section for improving performance.

#### 60.2.2.1 Tune the View Object SQL Statement

Tune both the list of attributes included in the view object as well as the underlying SQL statement. Avoid using the "one-size fits all" view objects, which include many other attributes that are not needed for your usage. These additional attributes consume unnecessary memory.

Capture the SQLs the view object is generating, with relevant view criteria applied, by enabling Java Business Objects (JBO) debug logging. (For expert-mode view objects, you should capture the SQL that you are providing). You can also generate explain
plans against a volume database to ensure performance is optimal and the correct indexes are in place.

If you must use hints to get a desirable execution plan for your query, set the **Query Optimizer Hint** field in the **View Object Editor - Tuning** section as shown in the following figure.

**Figure 60-4  View Object Editor — Tuning**

For user interface (UI) driven queries, the **FIRST_ROWS(10)** hint is necessary to instruct the optimizer to pick a plan that is optimized to return the first set of rows. Set this hint for view objects that are used for UI components, which typically just display the initial set of rows (such as table). If you are fetching all rows, then do not use the **FIRST_ROWS** hint.

### 60.2.2.2 Select the Correct Usage for View Objects

To maximize view object performance, the view object should match the intended usage. For more information about correct usage for view objects, see the "Creating View Objects" section in the **Tuning Performance**.

### 60.2.2.3 Set Appropriate Fetch Size and Max Fetch Size

How the view object is configured to fetch data plays a large role in the view object performance.

Due to the memory requirements for large batch size, we do not recommend using a fetch size of over 100. For view objects used on UIs, fetch size should not exceed 30.

---

**Caution:**

Java Database Connectivity (JDBC) pre-allocates memory to hold return data based on fetch size, so avoid the practice of applying a fixed fetch size, such as 30, to all view objects.

---

If you have a view object that is used in both query and insert, then call `setMaxFetchSize(0)` programmatically when you know it is being used in insert mode. In this case, you need to unset it when using it in query mode. You cannot set the **No Rows** option because the same view object is used in both insert and query mode in the same application module.

For view objects used for the List of Values (LOV) combo box, the number of rows fetched by Oracle ADF is controlled by the **ListRangeSize** setting in the LOV definition. The default value is 10 and a fetch size of 11 is appropriate. You should modify the value to 11.
For LOV text output, Oracle ADF fetches about 31 rows in the LOV search results window. To simplify retrieval, a fetch size of 11 is acceptable, to make it the same fetch size as the view object used in the LOV combo box. In this case, the data comes back in three round-trips, which is also acceptable.

**Note:**

ADF Business Components only recognizes fetch size if the SQL flavor is Oracle, which is what you should be using.

Fetch size can be set based on the usage of the view object. This is the appropriate place to set the fetch size for view objects that are used in different scenarios and the fetch size cannot be predetermined when the view object is created. You can edit the setting per view object usage by selecting the view object in the **Data Model** panel of the **Application Module editor**, clicking **Edit**, and then selecting the **Tuning** panel.

Fetch size can also be set at the view accessor level. This is typically used by teams consuming public view objects from other teams, such as for LOVs. The producer team would likely not set a fetch size since they cannot anticipate how their public view object would be used.

The following documents provide additional information related to subjects discussed in this section:

- For more information about tuning the *fetch* options for the application, see the "Configuring View Object Data Fetching" section in the *Tuning Performance*.

- For more information about list of values performance, see the "Working with List of Values (LOV) in View Object Attributes" section in the *Developing Fusion Web Applications with Oracle Application Development Framework*.

### 60.2.2.4 Use Bind Variables

Always use bind variables when setting the *WHERE* clause or when defining view criteria, as this allows the SQLs to be shared. However, there are some limited cases where you cannot use bind variables, such as when you need to use histograms. For more information, see "Additional View Object Configurations" in the *Tuning Performance* and "Working with Bind Variables" in the *Developing Fusion Web Applications with Oracle Application Development Framework*.

### 60.2.2.5 Include at Least One Required or Selectively Required View Criteria Item

When creating view criteria, include at least one view criteria item that is required, or selectively required, to use a database index and avoid a full table scan. Otherwise, the SQL generated will be of the form:

```sql
((MyTableColumn_name = :bvOrgId) OR (:bvOrgId IS NULL))
```

In this example, the query cannot be derived from an index on `MyTable.Column_name` due to the presence of the `:bvOrgId IS NULL` condition.

**Note:**

The `:bind IS NULL` condition is generated only if the View Criteria Item (VCI) is against a bind variable and the **Ignore Null Values** option is selected.
60.2.2.6 Use Forward-Only Mode when Possible

If a dataset is only traversed going forward, then forward-only mode can help performance when iterating through the dataset. The setForwardOnly API is defined on the RowSet interface, which ViewObjectImpl implements, so you can use it on secondary rowsets that you create via ViewObjectImpl.createRowSet(String name) as well. For more information about forward-only mode, see the "Configuring View Object Data Fetching" section in the Tuning Performance.

60.2.2.7 Avoid Calling getRowCount

Calling getRowCount on a view object results in all rows being fetched into memory. Unless you intend to actually fetch all the rows, avoid this call. Use a combination of vo.hasNext, hasPrevious, getCurrent, or vo.getFetchedRowCount, if the row set has been executed and you are attempting to see if there is at least one row fetched.

If you really need to find out how many rows are in the result set, and you know the result set is likely going to contain more than 50 rows, you should use getEstimatedRowCount. This triggers a count query but does not fetch all of the matching rows into memory.

There is also a method on the view object, getCappedRowCount(n), which executes a query and a count up to n number of rows. If the row count is less than n, it returns a positive number, and it returns a negative number if row count is more than n.

60.2.2.8 Avoid Entity Object Fault-in by Selecting Necessary Attributes Up-Front

If your view object is based on entity objects, and you request an attribute that is not fetched in the initial view object query, ADF Business Components must execute a "fault-in" SQL to fetch the entire entity object. This is expensive and can be avoided by initially selecting the list of attributes you are fetching in a view object. For example, if you know your validation logic accesses an attribute that is not displayed in the UI, fetch it in the initial view object query.

By default, only the key attributes are selected when executing a Declarative view object programmatically. A "fault-in" query is executed to get the rest of the attributes if they are referenced. To avoid this, use the following ViewObjectImpl methods to specify the columns that need to be selected when executing a Declarative view object programmatically: resetSelectedAttributeDefs, selectAttributeDefs, and unselectAttributeDefs.

If your view object is a declarative-mode view object that is accessed using view accessor, you may still be able to specify which attributes are accessed. If the code looks like the following example, the first line calls the accessor getter. The query is not executed until hasNext() is called on the last line. Therefore, you can use selectAttributeDefs to control which attributes are select after calling the getter and before calling hasNext.

RowSet rowSet = this.getPartySitePartySiteUseVVO();
RowSetIterator iterator = rowSet.createRowSetIterator("sellToUseIterator");
iterator.reset();
Row row = null;
if (iterator.hasNext()) {

60.2.2.9 Reduce the Number of View Object Key Attributes to a Minimum

If your view object is based on multiple entity objects, restrict the number of Key Attributes to a minimal set that uniquely identifies the row.
Note:
By default, the primary key of the view object is the concatenation of the primary key of all the underlying entity objects, which typically will be a lot more columns than what is actually needed to uniquely identify a row.

For those attributes that do not need to be part of the key, deselect the Key Attribute option in the View Object Attribute Editor. However, ensure that you include sufficient attributes. If your view object is updateable, comply with File.AdfModel.112.

### 60.2.2.10 Use Range Paging when Jumping to Different Row Ranges

View objects provide a mechanism to page through large datasets, giving users the ability to jump to a specific page in the results. To implement this feature, select Range Paging Incremental from the Access Mode dropdown list in the View Object Editor - Tuning section, as shown in the following figure.

**Figure 60-5  View Object Editor — Tuning: Access Mode**

For more information, see the "Optimize large data sets" row in the "Additional View Object Configurations" table in the Tuning Performance.

### 60.2.2.11 Use setListenToEntityEvents(false) for Non-UI Scenarios

The setListenToEntityEvents(false) method instructs the view object not to listen to entity events and therefore, the view object and all its row sets do not receive events generated from changes to entity row data. This is useful for batch processing because suppressing events improves performance.

Note:
These events are not related to the business events that you may have defined in the entity object.

When you call an association accessor, an internal view object is created. If you insert or update via the association accessor, you can call setListenToEntityEvents(false) for the internal view object by casting it to a RowSet as shown in the following example.

```java
RowSet myRowSet = (RowSet) myEntityImpl.getAttribute("<Accessor Name>");
((ViewObjectImpl) MyRowSet.getViewObject()).setListenToEntityEvents(false);
```

### 60.2.2.12 Use Appropriate Getter or Setter on View Row

If you have a ViewRowImpl class generated for your view object, call the named getter or setter if possible. For example, getEmployeeName or setEmployeeName, rather than the generic getAttribute or setAttribute.
Note:

Performance alone is not a sufficient reason for creating a custom `ViewRowImpl` class.

If you must use the generic `getAttribute` or `setAttribute`, consider using the index instead of the name for a small performance gain. It may be more troublesome to maintain the numeric attribute indexes, but for cases where you are looping through a large number of attributes, consider using `getAttribute(int index)` and `setAttribute(int index)`.

60.2.2.13 Use Appropriate Indexes with Case-Insensitive View Criteria Items

A view criteria item on a `varchar2` column is, by default, marked as case-insensitive. The generated predicate is in the form of `UPPER (column_name) operator UPPER (:bindVariable)`. Since the left-hand side is `UPPER (column_name)`, the existing non-function-based indexes created based on `column_name` is of no use for this kind of clause, and consequently, expensive table scans can result if this view criteria item is supposed to be the driving filter. Ensure there are appropriate function indexes to support case-insensitive searches.

60.2.2.14 Avoid View Object Leaks

In general, avoid creating a view object at runtime. Instead, add the view object instance to the application module and let the framework create it for you. If you have a use case where you must call `createViewObject` to create the view object, explicitly give it a name and first check if a view object with that name already exists in the application module. If it is already there, reuse it rather than create another one. If you no longer intend to use a dynamically created view object, remove it from the application module to avoid memory leaks.

60.2.2.15 Provide a "Smart" Filter when Using LOV Combobox

When you define a Combo Box with List of Values, provide an additional view criteria using the `Filter Combo Box Using` option so that the user only sees a list of frequently used choices. It typically does not meet business needs to return something like the first 10 customers in the system.

60.2.2.16 Use Small ListRangeSize for LOVs

When you define a LOV for a view object attribute, there is a `ListRangeSize` property (visible only in source), which defaults to 10. This controls the number of values to fetch when the combo box is selected on the UI. Do not change the `ListRangeSize` to a large value. In particular, -1 should never be used as it brings back all the rows.

60.2.2.17 Avoid Reference Entity Objects when not Needed

If your view object includes reference entity objects, they are loaded via separate queries whenever the key column values are changed. Therefore, if you have a scenario where attributes from the reference entity objects are not needed, use a view object that does not include reference entity objects. An example of this is when you are programmatically inserting rows and the reference entity object attributes do not need to be shown.
60.2.2.18 Do Not Use the "All at Once" Fetch Mode in View Objects

If you select the "All at Once" view object fetch mode, the view object query returns all rows, even though you are looking at only the first row. Depending on the query, this could cause OutOfMemory errors as the result of too many rows being fetched. Use the default "As Needed" fetch mode instead.

60.2.2.19 Do Not Use the "Query List Automatically" List of Value Setting

If you use the "Query List Automatically" option in the UI hints panel in the edit LOV screen, a query is executed by default, which could be expensive. This setting impacts only whether a search is executed by default when the LOV search list displays. For LOV combo boxes, regardless of this setting, the smart filter executes when the LOV combo is clicked and the dropdown list displays.

60.2.2.20 Avoid the "CONTAINS" or "ENDSWITH" Operator for Required or Selectively Required View Criteria Items

Required or Selectively Required view criteria items should use indexes so that their queries are efficient. If you use the "CONTAINS" or the "ENDSWITH" operator on a view criteria, the indexes cannot be used efficiently, resulting in poor query performance. Use an "Equals" or "Starts With" operator instead.

60.2.3 Work with Application Modules

When working with application modules, consider the suggestions in this section for improving performance.

60.2.3.1 Enable Lazy Delivery

When the Lazy Delivery option is enabled, ADF Business Components defers the creation of view object instances and nested application modules until they are requested. For more information, see the "Data Delivery - Lazy versus Immediate" section in the Tuning Performance.

60.2.3.2 Make Application Code Passivation-Safe

An application module is an ADF Business Components container that encapsulates business service methods and an active data model for a logical unit of work related to an end-user task. It is a wrapper for view objects and entity objects in a business model, handles all database transactions, and performs custom tasks by invoking application module specific service methods. For an ADF Business Components based web application, an HTTP request related to data operation cannot be processed without involvement of an application module instance. The following figure illustrates the application module position in the Oracle ADF applications architecture.

Figure 60-6    Oracle ADF Applications Architecture — Application Module Functions
Application module state management and application module pooling are important features provided out of the box by Oracle ADF. The combination of application module state management and pooling makes ADF Business Components based web applications more scalable by multiplexing application module instances in pool to serve large volume concurrent HTTP user sessions, and more reliable (failover) by serializing pending user session state into persistent storage (Database or File system).

Passivation is the process of serializing current states of an active application module instance to persist it to make it **passive**. Activation is its reverse process to activate a **passive** application module instance.

After coding and debugging all functional issues of an ADF Business Components based application, disable application module pooling to test and verify that the application code is **passivation-safe**. Disabling application module pooling enforces the application module instance to be released at the end of each request and be immediately removed (destructed), and passivation is triggered before its removal. On subsequent requests made by the same user session, a new application module instance must be created to handle this user request. A pending state must be restored from the passivation storage to activate this application module instance.

To disable application pooling for all application module pools, add `-Djbo.ampool.doampooling=false` to the JVM options when you run your page. You can also disable application pooling for select application modules.

**To disable application pooling for select application modules:**

1. Launch Oracle JDeveloper.
2. Right-click **Application Module** and select **Configurations**.
3. Click **Edit**, and select the **Pooling and Scalability** tab, as depicted in the following figure.
4. Deselect **Enable Application Module Pooling**.

![Figure 60-7 Application Module: Configurations — Pooling and Scalability](image)

The following documents provide additional information related to subjects discussed in this section:
• For more information about application module state management, see the "Application State Management" chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

• For more information about application module pooling, see the "Tuning Application Module Pools and Connection Pools" chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

60.2.3.3 Avoid Passivating Read-Only View Objects

There is performance overhead associated with passivation and activation. It is important to know of cases of where not to use this feature without impacting scalability and reliability. For example, there usually is no need to passivate view objects used in validators or helper functions. Similarly, if you have View objects where none of the attribute values are used across requests, such as view objects used only in service calls, then disable passivation.

To disable passivation for a view object, uncheck the Passivate State option in the View Object Editor, as shown in the following figure.

Figure 60-8  View Object Editor — Tuning

60.2.3.4 Avoid Passivating Certain Transient Attributes of a View Object

In addition to read-only view objects, some transient values of a view object, including transient view object attribute and calculated view object attribute, are read-only or their values are derived from other attributes via getter or groovy logic. There is no need passivate them.

• A transient view object attribute is an attribute which is not mapped to a table column or SQL calculation value, but its value is provided by Accessor function code.

• A calculated view object attribute is an attribute which is not mapped to a table column but is a SQL calculation expression.

To disable passivation for a subset of view objects' transient values, deselect Include All Transient Values in the View Object Editor - Tuning section, as shown in Figure 60-8. Then check the Passivate check box only for the attributes that require passivation in the view object attribute editor.

For more information, see "Managing the State of View Objects" in the Developing Fusion Web Applications with Oracle Application Development Framework.
60.2.3.5 Maintain Application Session User Tables

By default, Oracle ADF takes care of passivating all necessary states of an application module instance, but some custom information must be addressed by application code. Some examples of custom information are:

- Private member fields in view objects, entity objects, or application modules.
- User session state cached in Application Session UserData hash table. Go through:

  ApplicationModule.getDBTransaction().getSession().getUserData()

Caution:

This is not the user session. This is the Application Module session that ties to an application module and is maintained by ADF Business Components.

It is easy to confuse an Application Module session object with an HttpSession object, which also provides a hash table to cache some session user information. The difference is the HttpSession exists at the ADF Controller layer and the state cached in it can be across HTTP requests independent of the application module instance. However, an Application Session exists at the ADF Model layer and is per application module instance, so state cached in it can not be across HTTP requests after the application module instance switching happens.

Avoid saving a lot of custom session states in HttpSession because it increases memory usage and impacts scalability. This is the exact problem that application module state management and application module pooling is expected to solve.

To handle custom session states, you need to override passivateState() and activateState() functions in your ApplicationModuleImpl class or relevant VOImpl class.

The following is sample code that shows one way from the Pathfinder Application to passivate and activate the UserLoginPrincipal object.

Note:

XML documents can only handle String. Therefore, an object must be serialized before saving.

```java
public void passivateState(Document doc, Element parent)
{
    super.passivateState(doc,parent);

    UserLoginPrincipal principal = (UserLoginPrincipal)
    ApplicationModule.getDBTransaction().getSession().getUserData().get(Constants.LOGIN_PRINCIPAL);

    ByteArrayOutputStream baos=new ByteArrayOutputStream();
    try{
        ObjectOutputStream oos = new ObjectOutputStream(baos);
        oos.writeObject(principal);
        oos.flush();
    } catch(IOException e) {}  
}
```
String strPrincipal = baos.toString();

Node node = doc.createElement(Constants.LOGIN_PRINCIPAL); // Login should easily be converted into String
Node cNode = doc.createCDATASection(Constants.LOGIN_PRINCIPAL);
cNode.setNodeValue(strPrincipal);
node.appendChild(cNode);
parent.appendChild(node);

public void activateState(Element elem) {
    super.activateState(elem);
    if (elem != null) {
        NodeList nl = elem.getElementsByTagName(Constants.LOGIN_PRINCIPAL); // no idea what tags can be used
        if (nl != null) {
            for (int i=0, length = nl.getLength(); i < length; i++) {
                Node child = nl.item(i).getFirstChild();
                if (child != null) {
                    String strPrincipal = (String)child.getNodeValue();
                    ByteArrayInputStream bais = new ByteArrayInputStream(strPrincipal.getBytes());
                    ObjectInputStream ois = null;
                    try {
                        ois = new ObjectInputStream(bais);
                        principal = (UserLoginPrincipal)ois.readObject();
                    } catch (IOException e) {
                    } catch (ClassNotFoundException e) {
                    }
                    getSession().getUserData().put(Constants.LOGIN_PRINCIPAL,principal);
                    break;
                }
            }
        }
    }
}

For more information about how to manage custom user information, see the "Application State Management" chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

60.2.3.6 Tune the Application Module Release Level

The default release level is Managed, which implies that the application module's state is relevant and has to be preserved for this data control to span over several HTTP requests. In some cases you can programmatically set the release level to Unmanaged ("Stateless") at run time for particular pages to achieve better performance (no passivation). A classic example is the Logout page. Usually you can programmatically
release the application module with the unmanaged level when you want to signal that the user has ended a logical unit of work.

**Caution:**

When using `DCDataControl::resetState()` to set an *Unmanaged* release level, it only affects the current application module instance in the current request. For the next request, the application module instance automatically uses the default *Managed* release level again.

Setting the release level to *Reserved* makes Data Control "sticky" to an application module instance and all requests from the same `HttpServletRequest` associated with this Data Control are served by the same application module instance. This is contrary to the initiative of introducing application module state management and application module pooling, so using this release level is strongly discouraged.

**Caution:**

When the release level is changed to *Reserved* by calling `DCJboDataControl::setReleaseLevel()` with input argument `ApplicationModule.RELEASE_LEVEL_RESERVED`, it stays at this level until explicitly changed.

The following table illustrates application module release mode comparisons.
<table>
<thead>
<tr>
<th>Release Mode</th>
<th>Unmanaged (Stateless)</th>
<th>Managed (Stateful)</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Module Behavior</td>
<td>Does not preserve the state of the application module instance between page-processing requests. The instance is immediately released when a JavaServer Page (JSP) page terminates. <strong>Note:</strong> Select this option when you expect many users to access your JSP application simultaneously. The stateless option allows more users to access a JSP application simultaneously at the cost of requiring the user to reconnect to a new application module instance every time a JSP page is invoked (or re-invoked).</td>
<td>Preserves the application module instance’s state in the database between page-processing requests. This permits the application to maintain a user’s data without involving a single application module instance for long periods of time. <strong>Note:</strong> Stateful mode provides failover support for the HTTP session and is the preferred choice when the application module uses a standard JDBC connection.</td>
<td>Allocates the application module instance for the duration of the browser session. The instance is released only at the end of the session. This mode is provided primarily for compatibility with certain application module definitions. Failover is not supported in this mode. <strong>Note:</strong> Reserved mode is primarily useful when the application module requires a non-standard JDBC connection definition. Failover is not supported in this mode.</td>
</tr>
<tr>
<td>Release Mode &amp; User Action</td>
<td>Unmanaged (Stateless)</td>
<td>Managed (Stateful)</td>
<td>Reserved</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------</td>
<td>--------------------</td>
<td>----------</td>
</tr>
<tr>
<td><strong>DBTransaction</strong></td>
<td>Oracle ADF automatically posts and commits any changes because the application module state is not maintained between requests in stateless mode. The user is not expected to initiate the commit in stateless mode: the <strong>Commit</strong> and <strong>Rollback</strong> buttons are disabled in the JSP page.</td>
<td>Oracle ADF merely saves the application module state, including the data changes, to the database at the end of a page request. In this mode, the user is expected to initiate the commit by clicking the <strong>Commit</strong> button in the process JSP page. When the user clicks the <strong>Commit</strong> button, Oracle ADF immediately initiates a post and commit (together, as one step) on the database. Optionally, the user can click the <strong>Rollback</strong> button to prevent their changes from entering the database without ever initiating a post. Because the application module state is preserved, the user can initiate the Commit or Rollback at any point during the HTTP session.</td>
<td>Oracle ADF automatically posts any changes to the database (and initiates DML-specified database triggers on the effected tables). In this mode, the user is expected to click either the <strong>Commit</strong> button or <strong>Rollback</strong> button in the process JSP page. Because the application module itself is not released for the duration of the HTTP session, the user can initiate the Commit or Rollback at any point.</td>
</tr>
<tr>
<td><strong>Application Module Locking Behavior</strong></td>
<td>In stateless mode, the Business Components property <code>jbo.locking.mode</code> should be set to <code>optimistic</code>. Pessimistic locking is not compatible with stateless mode because the database transaction is always rolled back to allow the connection to be reused by another user. This results in the lock being released and makes pessimistic locking unusable.</td>
<td>In stateful mode, the Business Components property <code>jbo.locking.mode</code> should be set to <code>optimistic</code>. Pessimistic locking is not compatible with stateful mode because after the application module is preserved, the database transaction is rolled back to allow the connection to be reused by another user. This results in the lock being released and makes pessimistic locking unusable.</td>
<td>In release mode, you can reliably use pessimistic locking and may set the property <code>jbo.locking.mode</code> to <code>pessimistic</code>.</td>
</tr>
</tbody>
</table>
For more information about application module release level and state management, see the "Application State Management" chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

60.2.3.7 Do Not Leave Uncommitted Database Updates Across Requests

If you make database updates during a request, using either a DBTransactionImpl.pst changes call or PLSQL, and ensure the changes are committed within the same request, or rolled back if there are errors. All exceptions must be caught and rolled back to prevent partial updates from lingering in the database.

60.2.3.8 Release Dynamically Created Root Application Modules

If you create an application module using createRootApplicationModule calls, call the releaseRootApplicationModule to avoid a memory leak. Oracle ADF internally maintains references to these application modules, so they are not freed until you release them. You must also call releaseRootApplicationModule if you call one of the *AMIImpl.getInstance calls for the various Applcore application modules.

60.2.3.9 Do Not Destroy the Application Module when Calling Configuration.releaseRootApplicationModule.

Call Configuration.releaseRootApplicationModule(am, false) instead of Configuration.releaseRootApplicationModule(am, true). If true is passed, the application module is destroyed and the next request for this application module will be expensive because it must be created. If false is passed, the application module is released back to the application module pool and the next request can simply check out the application module from the pool, thereby avoiding the creation cost.

60.2.4 Work with Services

When working with services, consider the following suggestions for improving performance.

60.2.4.1 Set the Find Criteria to Fetch Only Attributes that are Needed

By default, when you call the find service, the child service data objects are also fetched. If you do not need those children, set the find criteria to fetch only the attributes you need.

The following example is sample code showing how to create a find criteria.

```java
FindCriteria fc = (FindCriteria)DataFactory.INSTANCE.create(FindCriteria.class);
//create the view criteria item
List value = new ArrayList();
value.add(new Integer(10));
ViewCriteriaItem vci = (ViewCriteriaItem)DataFactory.INSTANCE.create(ViewCriteriaItem.class);
vci.setAttribute("Deptno");
List<ViewCriteriaItem> items = new ArrayList(1);
items.add(vci);
//create view criteria row
ViewCriteriaRow vcr = (ViewCriteriaRow)DataFactory.INSTANCE.create(ViewCriteriaRow.class);
vcr.setItem(items);
//create the view criteria
List group = new ArrayList();
```
group.add(vcr);
ViewCriteria vc = (ViewCriteria)DataFactory.INSTANCE.create(ViewCriteria.class);
vc.setGroup(group);
// set filter
fc.setFilter(vc);

List cfcl = new ArrayList();
ChildFindCriteria cfc = (ChildFindCriteria)DataFactory.INSTANCE.create(ChildFindCriteria.class);
cfc.setChildAttrName("Emp");
cfc.setFetchStart(1);
cfc.setFetchSize(1);
cfcl.add(cfc);
fcl.setChildFindCriteria(cfcl);
DeptResult dres = svc.fndDept(fc, null);
pw.println("### Dept 10 and 2nd Emp ###");

60.2.4.2 Expose Service for Frequently Used Logical Entities
If you are doing frequent fetches of a business entity that is not the top level of a business object, it is better to expose a find service for that business entity rather than expose a find service for the highest level. Otherwise, the service call must be made against the topmost level entity, incurring unnecessary cost.

60.2.4.3 Use Correct ChangeOperation when Calling a Service
When you are using the processXXX() method to insert new rows, call the processXXX() method using ChangeOperation.CREATE. Do not call the processXXX() method with ChangeOperation.MERGE because it issues extra queries to the database to check if the rows already exist.

60.2.4.4 Set Only Changed Columns on Service Data Objects for Update
When creating a list of service data objects to pass for update using the processXXX() method, set only the columns that you really need to change, if possible. Service data objects with fewer attributes that have been set are updated faster than service data objects with all the attributes set.

60.3 ADF ViewController Layer Guidelines
Follow the best practices described in this section while working with various ADF ViewController layer components, such as geometry management, page templates, and partial page refresh.

60.3.1 Work with Various ADF ViewController Components
When working with ADF ViewController components, consider the suggestions in this section for improving performance.

60.3.1.1 Minimize the Number of Application Module Data Controls
For a specific page or page fragment, try to use only one application module data control. Use nested application modules rather than a separate application module data control to minimize the number of database connections your page uses. When using a nested application module, be sure to drag the nested application module from under the root application module in the data control panel.
**Note:**
If you use nested application modules, you can not pull data from different databases.

### 60.3.1.2 Use the Visible and Rendered Attributes

All ADF Faces Rich Client display components have two properties that relate to whether the component is displayed on the page. For more information about how to use these properties, see "ADF Faces Component Attributes" in the Tuning Performance.

### 60.3.1.3 Remove Unused Items from Page Bindings

If you decide to remove an unused item, such as a column from a table, remove the corresponding item from the tree binding. If you forget to do so for an expensive computed attribute, the logic to compute the attribute still executes even after removing the computed attribute from the table. In addition, remove any unused iterator bindings from the page definition file.

### 60.3.1.4 Disable Column Stretching

Columns in the `table` and `treeTable` components can be stretched so that there is no unused space between the end of the last column and the edge of the `table` or `treeTable` component. This feature is turned off by default due to high performance impact. Turning this feature on has a performance impact on the client rendering time so it should not be used for complex tables.

### 60.3.1.5 Use Appropriate Values for Refresh and RefreshCondition

The default values of the `Refresh` and `RefreshCondition` properties of iterator binding and action binding are *deferred* and *NULL*, which means that the related action binding will be invoked only if needed. The default value is appropriate for most cases. If you select the *ifNeeded* value for the `Refresh` property, the iterator or action may get refreshed twice and impact performance. The following figure shows how the JavaServer Faces (JSF) and Oracle ADF phases integrate in the lifecycle of a page request.
In particular, the value *always* should not be used as the Refresh property for `invokeAction` bindings. Using *ifNeeded* is the best option for most cases. Note that if `invokeAction` binds to the `methodAction`, which does not accept any parameters, or to any action, then it fires twice per request. To avoid this situation, use the `RefreshCondition` attribute on `invokeAction` to control when the action needs to fire.

For more information about the Refresh property, see the *What You May Need to Know About Using the Refresh Property Correctly* section in the "Understanding the Fusion Page Lifecycle" chapter of the *Developing Fusion Web Applications with Oracle Application Development Framework*.

**60.3.1.6 Disable Estimated Row Count if Necessary**

In addition to the query used to fetch the data for display, Oracle ADF issues a count query to calculate the estimated result set size for view objects that are bound to a table on the UI. This is used to size the scroll bar and is capped at a certain threshold to avoid scanning the entire result set. If your query is expensive and this additional query results in your page not meeting your performance target, consider setting the `RowCountThreshold` setting to a negative value, to turn off the row count query.

You can consider disabling row count completely by setting `RowCountThreshold` to `-1` after extensive tuning. Then you could apply the global `RowCountThreshold` if your count query still has performance issues.

---

**Caution:**

When you disable the estimated row count, the scrolling behavior of your table is different. The user can scroll forward only one range at a time.

---

**60.3.1.7 Use HTTPSession Hash Table in Moderation**

`HTTPSession` provides a hash table to cache user information. However, all the information is saved in memory, so inappropriate use of `HTTPSession` cache causes some scalability issues, including:
- High memory usage on the server
- User information loss if the server is down
- Increased network traffic to replicate session state in a clustered environment

Putting critical, large volume information in `HttpServletRequest` cache is not recommended. Instead, you should leverage application module state management and application module pooling. See Make Application Code Passivation-Safe.

The following example shows how to use `HttpServletRequest` cache in a backing bean.

```java
((HttpServletRequest)
  (FacesContext.getCurrentInstance().getExternalContext().getRequest()))
getSession().setAttribute("UserLoginPrincipal", sessionLoginPrincipal);
```

### 60.3.1.8 Use Short Component IDs

Sometimes you must provide an ID for a UI component. For example, an ID is required for a component that is a source of a partial page refresh (PPR) event. Also, Oracle ADF generates default component IDs for certain components, such as when a task flow is added to a page as a region. (The default region ID is the first 5 characters of the task flow name plus a digit). If you have pages with a region ID that is greater than 7 characters, you should shorten the IDs of the task flow regions to 7 characters or fewer (including the digit), with 3 characters being ideal.

If IDs are specified for other naming containers (such as tables), a length of 3 or fewer is best. Using short naming for container IDs helps to reduce the size of each response, as well as network traffic, because the IDs of the parent naming containers are appended to a child's generated ID.

### 60.3.1.9 Follow UI Standards when Using Search

When using Search, follow these UI standards:

- Blind queries are not allowed.
- Use Match All instead of Match Any when there are multiple criteria.

### 60.3.1.10 Do not set Client Component Property to True

ADF Rich Client has a sparse component tree on the client. This means only required components are created on the client. The component instance is instantiated on the client if:

- Client component property is true by default. For example, as required by Oracle ADF
- Client-side event listener is registered, which you should not be using
- Due to needed client-side interaction with component, the client component property is set to true

To achieve the best performance, do not set the client component property to true.

### 60.3.1.11 Set Immediate Property to True when Appropriate

ADF Rich Client components have an immediate attribute. There are some cases where setting immediate to true can lead to better performance. For more information, see the "ADF Faces Component Attributes" section in the Tuning Performance.
60.3.1.12 Use Appropriate Content Delivery Mode for a Table or a Tree Table

By default, the data for Table, Tree and other stamped components uses the lazy data delivery mechanism. This means that page content is delivered with the first response from the server and the next request fetches the data. This option should be used when the page has enough content to be displayed and a table query may be slow. Underneath, the data fetch request uses the table streaming feature, which delivers table data to the client as soon as it is available. Also, it provides the ability to execute data fetch requests on the server in parallel, making them faster. To enable fetching data in parallel, set the RenderHint property of the iterator to background. This option could increase the number of database connections.

The other option to deliver data is immediate mode, which is set on the table. In this mode, the data is delivered with the initial page. This is better in terms of CPU and memory consumption on the server, and should be used if the table is the main context of the page.

For more information, see "Data Delivery - Lazy versus Immediate" in the Tuning Performance.

60.3.1.13 Set the Appropriate Fetch Size for a Table

Tables have a fetch size which defines the number of rows to be sent to the client in one round-trip. To get the best performance, keep this number low while still allowing enough rows to fulfill the initial table view port. This ensures the best performance while eliminating extra server requests.

In addition, consider keeping the table fetch size and iterator range size in sync. By default, the table fetch size is set to the EL expression #{$bindings.<name>.rangeSize} and should be equal to the iterator size. The iterator range size should be set to number of displayed rows + 1. In particular, for auto-height tables, set the iterator range size to the value of autoHeightRows + 1.

60.3.1.14 Avoid Frozen Columns and Header Columns if Possible

Frozen columns and header columns in the table are very expensive on the client side and should be avoided if possible. Overhead can be 20% to over 100% for a simple page with limited content when there are frozen columns. If frozen columns must be used, ensure the row height of the columns to the left and the right of the frozen column are of the same height.

60.3.1.15 Avoid Unnecessary Regions

Regions are very powerful and provide extreme flexibility. However, there is an associated cost with every region. To have the best performance, use the region only when it is needed. Generally, the Oracle ADF guideline is to not have more than 15 regions on the page.

60.3.1.16 Set the Data Control Scope to "Shared"

Set the Data Control Scope to "Shared" for a task flow to allow sharing of the data control. (This is the default). This reduces the number of database connections. There may be some cases where using Isolated is functionally necessary. For more information, see the "Sharing Data Controls Between Task Flows" section in the Developing Fusion Web Applications with Oracle Application Development Framework.

If you use the same task flow multiple times on a page, and you are using shared data control scope, by default it may not work because the data controls will show the same data. To address this, instead of switching to isolated data control scope, follow the
60.3.1.17 Select the No Save Point Option on a Task Flow when Appropriate

Select the **No Save Point** option if you do not need the functionality to roll back changes at the end of the task flow. If you do not use this option, the model state is passivate at the beginning of the task flow, which is expensive.

60.3.1.18 Use Click-To-Edit Tables when Appropriate

For tables where most rows are usually view-only and rarely edited, set the **Edit** mode property to *Click-To-Edit*. This reduces the response size significantly and improves performance.

60.3.1.19 Avoid Unnecessary Task Flow Activation for Regions Under Popups

By default, task flows that use a region under popups are activated when the page loads, not when the popup displays. This causes unnecessary memory and CPU usage if the user does not use the popup. There are two approaches for activating the task flow region only when the popup displays:

1. Set the following properties to "deferred":
   - The **childCreation** property on the popup.
   - The activation property on the task flow binding. (This is under the **Executables** section in the page definition file.)

2. Set the activation property on the task flow binding to "conditional" and specify a condition in the "active" to an EL expression that returns true when the popup displays. Usually this requires creating a view scope variable that is set using a **setPropertyListener** executed on **popupFetch**. The EL expression must return true if the popup is displayed. (A request scope variable will not work in most cases unless you cannot submit any server requests from the popup.)

Approach (1) is simpler but you must use approach (2) for the following cases:

- Any of the following tags are present inside the popup attribute:
  - `f:attribute`
  - `af:setPropertyListener`
  - `af:clientListener`
  - `af:serverListener`

- You need to refer to any child components of the popup before the popup is displayed. Setting **childCreation="deferred"** postpones the creation of any child components of the popup and you cannot refer to them until after the popup displays.

60.3.1.20 Delay Creation of Popup Child Components

This recommendation is similar to **Avoid Unnecessary Task Flow Activation for Regions Under Popups**, but is applicable to popups that do not contain regions. By default, the child components under a popup are created even when the popup is not accessed. This causes unnecessary memory and CPU usage if the user does not use the...
popup. To avoid this overhead, set the childCreation property on the popup to "deferred".

This approach cannot be used for the following cases:

- Any of the following tags are present inside the popup attribute:
  - f:attribute
  - af:setPropertyListener
  - af:clientListener
  - af:serverListener
- You need to refer to any child components of the popup before the popup is displayed. Setting childCreation="deferred" postpones the creation of any child components of the popup and you cannot refer to them until after the popup displays.

60.3.1.21 Avoid Unnecessary Task Flow Activation for Regions Under Switchers

By default, task flows that use an \texttt{af:region} under switchers are activated regardless of whether the facet displays. This causes unnecessary memory and CPU usage for the facets that do not display. To activate the task flow region only when it displays, set the 'activation' property on the task flow binding to 'conditional', under the Executables section in the page definition file. Also specify a condition in the "active" to an EL expression that returns true when the facet displays.

Typically, you may already have an EL expression to control the return value for the facetName property in the switcher. For example, if your switcher looks like this:

```xml
<af:switcher id="s1" defaultFacet="1" facetName="#{pageFlowScope.facet}">
  <f:facet name="1">af:region value="#{bindings.TF1.regionModel}" id="r1"/>
  <f:facet name="2">af:region value="#{bindings.TF2.regionModel}" id="r2"/>
</af:switcher>
```

The associated binding should have activation set to "conditional", and active set to an EL, as follows:

```xml
<taskFlow id="tTF1" taskFlowId="<some task flow>" active="#{pageFlowScope.facet=='1'}" activation="conditional" xmlns="http://xmlns.oracle.com/adf/controller/binding"/>
<taskFlow id="tTF2" taskFlowId="<some other task flow>" active="#{pageFlowScope.facet=='2'}" activation="conditional" xmlns="http://xmlns.oracle.com/adf/controller/binding"/>
```

60.3.1.22 Avoid Unnecessary Root Application Module Creation from UI-layer Code

Creating additional root application modules is expensive when you can reuse the root application module that is associated with the data bindings on your page. For example, do not access an application module instance by calling the \texttt{Configuration.createRootApplicationModule()} API from a backing bean. This results in creating additional application module instances, which are distinct from the application module instance that is automatically checked out and in by the Oracle ADF data binding layer, used by UI pages and task flow method call activities. This can lead to performance and memory issues if your backing bean calls \texttt{Configuration.createRootApplicationModule()} API without calling \texttt{releaseRootApplicationModule()}. 
Use an ADFM action binding to invoke a client interface method declaratively on an application module instead. This approach requires no code and often prevents the need for a backing bean. It also ensures that any exceptions are handled in a consistent way as if Oracle ADF had invoked the method declaratively. Also ensure that your backing bean is invoked in a context where a pageDef has been defined.

The following code excerpt is an example that follows our recommendation:

```java
private ComponentReference<RichTable> allocationTableRef;
public void setAllocationTable(RichTable allocationTable) {
    if (this.allocationTableRef == null)
        this.allocationTableRef =
            ComponentReference.newUIComponentReference(allocationTable);
}
publi RichTable getAllocationTable() {
    return allocationTableRef==null ? null : allocationTableRef.getComponent();
}
```

The following example is not recommended:

```java
private RichTable allocationTable;
publi void setAllocationTable(RichTable allocationTable) {
    this.allocationTable = allocationTable; }
publi RichTable getAllocationTable() {
    return allocationTable; }
```

60.3.1.23 Avoid Unnecessary Savepoints on Task Flow Entry

When the transaction setting of a task flow is "Always Use Existing Transaction" or "Reuse Existing Transaction if Possible", and the "No savepoint on taskflow entry" box is not checked, Oracle ADF automatically creates a savepoint when entering the taskflow. Check the box to avoid the savepoint cost if you do not have functionality to rollback to this particular savepoint.

60.3.1.24 Cache Return Values in Backing Bean Getters

The common usage of backing beans is to reference values from EL expressions. Bean getters may also be called from other places in the code, as well as being called multiple times in a request. If you have expensive computations inside the bean getter logic, consider caching the results inside the bean. This is fairly safe to do for request-scope beans unless you expect the result to change within the request. For view-scope or page flow-scope beans, be careful about when to invalidate the cached results.

60.3.1.25 Do Not Maintain References to UI Components in Managed Beans

If you maintain direct references to UI Component instances from view scope or page flow scope beans, this could cause both functional errors and impact performance. If you must maintain a reference, use the ComponentReference pattern instead.

60.3.1.26 Set childCreation Property to Delay Child Component Creation

By default, child components for showDetailItem (under panelAccordion or panelTabbed), showDetailHeader, showDetail, panelBox, and showDetailFrame are created even if the parent is not disclosed. Set the childCreation property to "lazy" or "lazyUncached" to improve performance by postponing child component creation, including activating underlying task flow regions. Note that if the child component is a taskflow region, the activation property of the taskflow executable in the page definition must be set to "deferred" for this to work.
60.3.1.27 Delay Activation of Taskflow Regions That are not Always Rendered

If the parent component of a taskflow region has an EL expression on the rendered property, set the activation property of the taskflow executable in the page definition to "conditional". Set the active property to the same EL expression as the one that controls the rendered property. This setup avoids activating the taskflow unless the region is actually rendered.

60.3.2 Enable ADF Rich Client Geometry Management

ADF Rich Client supports geometry management of the browser layout where parent components are in the UI explicitly. The children components are sized to stretch and fill up available space in the browser. While this feature makes the UI look better, it has a cost. The impact is on the client side where the browser must spend time resizing the components. The components that have geometry management by default are:

- PanelAccordion
- PanelStretchLayout
- PanelTabbed
- BreadCrums
- NavigationPane
- PanelSplitter
- Toolbar
- Toolbox
- Table
- Train

Note the following recommendations for using geometry management:

- Consider using `panelGridLayout` to flatten HTML hierarchy and improve performance when designing page layouts for non-stamped component. `panelGridLayout` uses JavaScript to accomplish page layout like PSL, but it does not have the overhead of nesting JS layout components. To avoid performance degradation, do not use PanelGridLayout in stamped components.

- Try to minimize the number of child components that are under a parent geometry managed component.

- The cost of geometry management is directly related to the complexity of child components.

- The performance cost of geometry management can be smaller, as perceived by the user, for the pages with table or other data stamped components when table data streaming is used. The client-side geometry management can be executed while the browser is waiting for the data response from the server.

60.3.3 Use Page Templates

Page templates allow you to build reusable, data-bound templates that can be used as a shell for any page. You can build one or more templates that provide structure and
consistency for other developers building web pages. The templates have both static areas on them, that cannot be changed when they are used, and dynamic areas on them, where the developer can place content specific to the page they are building. Note the following important considerations when using templates:

- Since templates are present in every application page, they must be optimized so that common performance impacts are avoided. Adding round corners to the template, for example, can impact the performance for every page.

- When building complex templates, sometimes it is easier to build them in multiple pieces and include them in the top-level template using `<f:subview>` tag. However, from a performance perspective, this is not typically recommended since it can impact memory usage on the server side. `<f:subview>` introduces another level into the ID scoping hierarchy, which results in longer IDs. Long IDs have a negative impact on performance. Avoid using `<f:subview>` unless it is required. It is not necessary to use `<f:subview>` around `<jsp:include>` if you can ensure that all IDs are unique. For example, if you are using `<jsp:include>`, break a large page into multiple pieces for easier editing. Whenever possible, avoid using `<f:subview>`. If you are including content developed by someone else, use `<f:subview>` if you do not know which IDs the developer used. In addition, you do not have to put `<f:subview>` at the top of a region definition.

### 60.3.4 Use ADF Rich Client Partial Page Rendering (PPR)

ADF Rich Client is based on Asynchronous JavaScript and XML (Ajax) development technique. Ajax is a web development technique for creating interactive web applications, where web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, without the whole web page being reloaded. The effect is to improve a web page's interactivity, speed, and usability.

With ADF Faces, the feature that delivers the Ajax partial page refresh behavior is called partial page rendering (PPR). PPR enables small areas of a page to be refreshed without having to redraw the entire page. For example, an output component can display what a user has chosen or entered in an input component or a command link or button can cause another component on the page to be refreshed.

Two main Ajax patterns are implemented with partial page rendering (PPR): native component refresh and cross-component refresh. While the framework builds in native component refresh, cross-component refresh has to be done by developers in certain cases.

Cross-component refresh is implemented declaratively or programmatically by the application developer defining which components are to trigger a partial update and which other components are to act as partial listeners, and so be updated. Using cross-component refresh and implementing it correctly is one of the best ways to improve client-side response time. While designing the UI page always think about what should happen when the user clicks a command button.

Consider a typical situation in which a page includes an `af:inputText` component, an `af:commandButton` component, and an `af:outputText` component. When the user enters a value for the `af:inputText`, then clicks the `af:commandButton`, the input value is reflected in the `af:outputText`. Without PPR, clicking the `af:commandButton` triggers a full-page refresh. Using PPR, you can limit the scale of the refresh to only those components you want to refresh, in this case, the `af:outputText` component. Perform the following steps to achieve this:
• Set up the af:commandButton for partial submit by setting the partialSubmit attribute to true. Doing this causes the command component to start firing partial page requests each time it is clicked.

• Define which components are to be refreshed when the partial submit takes place. In this example, the af:outputText component, by setting the partialTriggers attribute for each of them to the id of the component, triggers the refresh. This means setting the partialTriggers attribute of the af:outputText component to give the id of the af:commandButton component.

The main reason that partial page rendering can significantly boost performance is because the full page refresh does not happen and the framework artifacts (such as ADF Rich Client JS library, and style sheets) are not reloaded and only a small part of the page is refreshed. In several cases, this means no extra data is fetched or no geometry management occurs.

The ADF Rich Client has shown that partial page rendering results in the best client-side performance. Besides the impact on the client side, server-side processing can be faster and can have better server-side throughput and scalability.

60.4 SOA Guidelines for Human Workflow and Approval Management Extensions

For best practices while working with Human Workflow and Approval Management extensions (AMX), see the Oracle Human Workflow Performance Tuning chapter in the Tuning Performance.

60.5 Oracle Fusion Middleware Extensions for Applications Guidelines

When working with application modules, consider the best practices in this section, related to using a nested service and releasing application modules returned from getInstance calls.

60.5.1 Use Profile.get to Get Profile Option Values

To get a profile option value, use

```java
oracle.apps.fnd.appcore.Profile.get(<Profile Option Name>)
```

This is optimized to first find the profile value in an internal cache, so it checks out an application module only if needed. Avoid calling ProfileServiceAM.getInstance as it checks out a ProfileService application module instance, which is expensive.

60.5.2 Release any Application Modules Returned from getInstance Calls

If you have no other option and must use getInstance to get an application module back, such as ProfileServiceAM.getInstance, you must release it to avoid a memory leak via a Configuration.releaseRootApplicationModule call, as shown in the following example.

```java
ProfileServiceAM profileService = null;

try {
    profileService = ProfileServiceAMImpl.getInstance();
    String value = profileService.getProfileValue("<ProfileOptionName>");
}```
60.5.3 Avoid Unnecessary Activation of Attachments Task Flow

When the attachments feature is used, it creates a new task flow in the page bindings, as shown in the following example:

```xml
<taskFlow id="attachmentRepositoryBrowseTaskFlow1"
  taskFlowId="#{backingBeanScope.AttachmentBean.taskFlowId}"

or:

<taskFlow id="attachmentRepositoryBrowseTaskFlow1"
  taskFlowId="/WEB-INF/oracle/apps/fnd/applcore/attachments/ui/attachments-docpicker-
  taskflow.xml"

This task flow is unnecessarily activated. To avoid this situation, navigate to the bindings tab of the page where the Attachments component was added. Select attachments task flow, attachmentRepositoryBrowseTaskFlow1, from the list of Executables. Set the following attributes in the property inspector under Common:

- activation="conditional"
- active="#{pageFlowScope.FND_ATTACHMENTS_LOAD_TF==true}"

60.5.4 Use Static APIs on Message Get Message Text

Use static APIs from oracle.apps.fnd.applcore.messages.Message to get message text. Avoid using MessageServiceAMImpl.getInstance, or calling createRootApplicationModule to get MessageServiceAM, as this results in checking out and initializing an instance of MessageServiceAM from the AM pool, which has a cost.

60.5.5 Set the Data Control Scope to Isolated for Page Level Item Nodes

If the data control scope is shared for task flows pointed to by certain item nodes, then the life span of these task flow data controls is tied to the parent, which is either Main TF or Regional TF in the UI shell. This scenario applies to those item nodes with taskType equal to "defaultMain", "dynamicMain", or "defaultRegional". This means that DC frame is not removed for the duration of the session, regardless of any navigation or closing tab. This is because the Main TF and Regional TF in the UI shell have the DC scope set to shared, due to the requirement to share CE between the regional and main areas.

If there is no requirement to share transactional data between the regional and main areas, then set dataControlScope="isolated" on the page level item node in the menu file. This recommendation assumes that the underlying task flows used in the regional area or the task menu already have data control scope set to isolated. Note that you should not change the data control scope on the task flow itself.

60.6 General Java Guidelines

When working with Java, consider these best practices in this section, related to Strings and StringBuilder, Collections, Synchronization, as well as other Java features.
60.6.1 Work with Strings and StringBuilder

When working with Strings and StringBuilder, consider the suggestions in this section for improving performance.

60.6.1.1 Use StringBuilder Rather than the String Concatenation Operator (+)

When doing String concatenations inside a loop, see if the operation can be moved outside of the loop. Frequently, the concatenation code is put inside the loop even though the value can never change there.

The String concatenation operator + involves the following:

- A new StringBuilder is created.
- The two arguments are added to it with append().
- The final result is converted back with toString().

This increases cost in both space and time, especially if you are appending more than one String. Consider using a StringBuilder directly instead.

StringBuilder was introduced in Java Development Kit (JDK) 1.5 and is more efficient than StringBuffer since the methods are not synchronized. When using StringBuilder (or StringBuffer), optimally size the StringBuilder instance based on the expected length of the elements you are appending to it. The default size of a StringBuilder is 16. When its capacity is exceeded, the JVM has to resize the StringBuilder, which is expensive. For example, instead of doing this:

```java
String key = granteeType + "::" + granteeKey;
```

Follow this example instead:

```java
String key = new StringBuilder(granteeType.length() + 1 +
    granteeKey.length()).append(granteeType).append("::").append(granteeKey)
    .toString();
```

The StringBuilder object is initialized with the correct capacity so it can hold all the appended strings it needs to resize its internal storage structure.

For the sake of simplicity, it is acceptable to do String concatenation using "+" for debug log messages, if you follow the logging standard and check the log level before constructing the log message.

Avoid unnecessary use of String.substring calls since it creates new String objects. For example, instead of doing this:

```java
if (formattedNumericValue.substring(0,1).equals("-")) negValue = true;
```

Follow this example instead:

```java
if (formattedNumericValue.charAt(0) == '-') negValue = true;
```

The hashCode method is another common place where you do String concatenation. The following example uses the hashCode implementation and requires String concatenation on every call.

```java
public int hashCode()
{
    // ...
}
```
h = new StringBuffer(len).append(resp).append(rapl).toString().hashCode();
return h;
}

The following example does not use String concatenation.

public int hashCode()
{
    ... ...
    h = 37*h + (int)(m_respID ^ (m_respID >>> 32));
    h = 37*h + (int)(m_respApplID ^ (m_respApplID >>> 32));
    return h;
}

---

**Note:**
This example was taken from the book *Effective Java*.

---

Plan carefully before deciding to concatenate Strings. There are often alternative ways to implement the intended logic without concatenation.

60.6.1.2 Check the Log Level Before Making a Logging Call

Always check the log level before make a logging call, otherwise, many objects may be constructed unnecessarily. For example, if logging is disabled, but your code still calls the logging API that passes in the log message, this concatenates several String objects together and the String concatenation is a waste of resources. The log message is constructed and passed into the logging API, and then discarded since logging is disabled. If you first check if the target log level is enabled, then the log message does not need to be created unless it is actually needed.

The amount of information that applications log depends on how the environment is configured and how the application code is instrumented. To maximize performance, do not set the logging level higher than the default INFO level logging. If the logging setting does not match the default level, reset the logging level to the default for best performance. Once the application and server logging levels are set appropriately, ensure that the debugging properties or other application level debugging flags are also set to appropriate levels or disabled. To avoid performance impacts, do not set log levels to levels that produce more diagnostic messages, including the FINE or TRACE levels.

60.6.1.3 Use Proper Logging APIs for Debug Logging

Use proper logging APIs, such as AppsLogger, instead of using System.out.println and System.err.println for debug logging. This way, log messages are properly formatted with the correct context information.

60.6.1.4 Lazy Instantiation

Avoid instantiating objects until they are needed by using lazy loading for application modules. In the Application Module tuning section, customize runtime instantiation behavior to use lazy loading. Lazy loading can also be set JVM-wide by adding the following JVM argument: -Djbo.load.components.lazily=true. For example, if you are coding a method to do String replacement, do not allocate a StringBuilder object to do the replacement until you have found a fragment that must be replaced.
60.6.2 Configure Collections

When working with collections, consider the following:

- Legacy collections (like Vector and Hashtable) are synchronized, whereas new collections (like ArrayList and HashMap) are unsynchronized, and must be wrapped via Collections.SynchronizedList or Collections.synchronizedMap if synchronization is desired. Do not use synchronized classes collections, including collections from java.util.concurrent package, that are not shared among threads.

- Do not use object collections for primitive data types. Use custom collection classes instead.

- Size a collection based on the number of elements it is intended to hold to avoid frequent reallocations and rehashing in case of hashtables or hashmaps.

For more information about collections, see "Configuring Garbage Collection" in the Tuning Performance.

60.6.3 Manage Synchronization

When working with synchronization methods, consider the following:

- Avoid synchronized methods if possible, because even with the latest versions of the JVM, there is still significant overhead.

- Bad candidates for synchronization are:
  - Read-only objects
  - Thread local objects

- Minimize the size of the synchronized block of code. For example, instead of synchronizing the entire method, it may be possible to synchronize only part of the method.

- In JDK 1.5, there is a new package, java.util.concurrent, that contains many classes designed to reduce contention. For example, java.util.concurrent.ConcurrentHashMap provides efficient read access while still maintaining synchronized write access. These new classes should be evaluated instead of simply using a Hashtable whenever synchronization is required.

60.6.4 Work with Other Java Features

When working with Java features, consider the following:

60.6.4.1 Avoid Autoboxing

Autoboxing is a feature introduced in JDK 1.5, which allows direct assignment of primitive types to the corresponding object type, such as int => Integer. Avoid using autoboxing in code that is called repeatedly, as shown in this example:

```
Integer myInteger = 1000;
```

The following example shows how the compiled code basically creates a new Integer object based on the integer value:
8: bipush 100
10: invokestatic #2;
   //Method
   java/lang/Integer.valueOf:(I)Ljava/lang/Integer;
13: astore_2

If this piece of code is called repeatedly, then each call creates one Integer object and could have adverse performance impact.

60.6.4.2 Do not use Exceptions for Code Path Execution

Exception object and snapshot of stack have to be created. This is expensive, especially for typical Oracle ADF applications, which have very deep execution stacks. For example, if your code needs to detect whether a certain object can be casted to a certain type, use instanceof instead of doing the cast and catching the exception. In other words, use instanceof instead of relying on ClassCastException.

60.6.4.3 Reuse Pattern Object for Regular Expression Matches

If using regular expression classes to match against a known pattern, create the Pattern object only once and reuse it for subsequent matches. Only the Matcher object must be created each time.

60.6.4.4 Avoid Repeated Calls to the same APIs that have Non-Trivial Costs

Avoided repeated calls to the same APIs that have non-trivial costs. Use a local variable to hold the result instead. For example, instead of:

```java
if (methodA() >= 0)
   return methodA() + methodB();
```

Use:

```java
int res = methodA();
If (res >= 0)
   return res + methodB();
```

60.6.4.5 Close Unused JDBC Statements to Avoid Memory Leaks

To avoid memory leaks, closed unused JDBC statements. The following example depicts a statement leak in java code.

```java
String s1 = "BEGIN FND_GRANTS_PKG.UPDATE_GRANT("+
   " :1, "+
   " :2, "+
   " :3, "+
   " :4, "+
   " :5);"+
   " END;";

setGrant = txn.createCallableStatement(s1,1);
setGrant.setInt(1,v);
setGrant.setString(2,guid);
setGrant.setDate(3,sd);
setGrant.setDate(4,edt);
setGrant.registerOutParameter(5,Types.VARCHAR);
setGrant.execute();

There should be a setGrant.close() call to close the statement.
```
For every call to `createStatement()`, `prepareStatement()`, `prepareCall()`, `createCallableStatement()`, or `createPreparedStatement()` there should be a `close()` to prevent memory leaks.

In the case of query execution, it is possible that the result set may be closed, but the underlying statement has not been closed, as shown in the following example.

```java
public static AppsCtxtFileInfo readAppsCtxtFile(Connection pCon,
        AppsCtxtFileInfo fileInfo, boolean update)
    {
        AppsCtxtFileInfo ret = null;
        try
        {
            String query = getReadAppsCtxtFileString(false, update);
            PreparedStatement stmt = pCon.prepareStatement(query);
            stmt.setString(1, fileInfo.getNodeName());
            stmt.setString(2, fileInfo.getPath());
            ResultSet rs = stmt.executeQuery();

            if (rs.next())
            {
                ret = getAppsCtxtFileInfoFromResultSet(rs);
            }
            rs.close();
        }
        catch(SQLException e)
        {
            Logger.println(e, Logger.DEV_LEVEL);
            if (update)
            {
                if (e.getErrorCode()== ROW_LOCKED_ERROR_CODE )
                    throw new RowLockedException();
            }
            else
                throw e;
        }
        return ret;
    }
```

In this case, the result set is being closed via `rs.close()`. However, the statement (stmt) has not been closed. For every statement opened, close it in the final block of a `try-catch`, as shown in the following example.

```java
try
{
    ...
    <open the statement>
    <process the statement>
    ...
} catch
{
    ...
    <process any exceptions>
    ...
}
finally {
...
try {
  <close the statement>
}
catch {
  <process any exceptions>
}
...
}

Make sure to catch exceptions around something like `stmt.execute()`.  

60.6.4.6 Use registerOutParameter to Specify Bind Types and Precisions  

In Java files, whenever a callable statement is fired, such as in a begin-end block, the out bind types and precision have to be specified. This is done after creating the callable statement but before the query is executed. The method call to specify the type is called `registerOutParameter()`. This call should exist for every out bind in the callable statement regardless of its return type. There are two overloaded versions of this method call that can be used:

```
registerOutParameter(int paramIndex, int sqlType, int scale, int maxLength)
registerOutParameter(int paramIndex, int sqlType)
```

60.6.4.7 Avoid JDBC Connection Leaks  

If you are getting a connection directly from Data Source or through `ApplSessionUtil.getConnection`, ensure you release the connection in a final block.

60.7 Cache Data  

Caching is one of the most common approaches for improving the performance of accessing commonly used data. Shared application modules and view objects provide a mechanism for storing database results and other objects, such as in-memory ADF Business Components objects for repeated usage. This minimizes expensive object initializations and database round-trips, which ultimately results in improved application performance.

60.7.1 Identifying Data to Cache  

It is important to correctly identify the best data to cache. Generally, this is the data that is common to different users, frequently used and infrequently changed, expensive to retrieve from the database or data source, and expensive to create. Data suitable for caching should have some or all of the following characteristics:

- **Shared Objects**: Data that is common across users is more appropriate than user specific data.
- **Long-Lived**: Data that is long-lived is more appropriate than short-lived data that is valid only for a user request.
• **Expensive to Retrieve**: Objects that take a long time to retrieve, such as objects that are obtained from expensive SQL queries, are good candidates.

• **Expensive to Create**: Objects that are frequently created or take a long time to create are appropriate. Frequent creation can be avoided by caching the instances.

• **Frequent Use**: Objects that have a high probability of being frequently used are appropriate. Caching objects that are not actively used needlessly occupy JVM memory.

• **Infrequently Changed**: The cached data is invalidated and removed from cache whenever it is changed, which makes caching frequently changed data more costly.

Lookup codes are an example of data that meets most of the above criteria.

### 60.7.2 How to Add Data to Cache

Cached objects are stored in view objects, which are added to an application module. This application module is configured as shared at the application level so that the cached objects are available to all users. To stripe cached data, view criteria with bind parameters are used. For example, a view accessor on top of a shared `DeptVO` with a view criteria such as `location=:bindLocation` results in one cache for each distinct value of `:bindLocation`.

Perform the following steps to add data to cache:

1. Create a view object or identify an existing view object to store the cached data.

2. (Optional) Create commonly used view criteria for the shared view object.

3. (Optional) Configure the shared view object's property, such as time to live, and so on.

4. Create one application module for each product that contains all the view objects to store the cached data.

5. Cache a short list of data by pre-loading all the data into memory. This prevents subsequent queries requiring additional database trips. To do this, generate the `VOImpl` class for the shared view object and override the `create()` function of `VOImpl` to fully populate the view object cache, as shown in the following example.

```java
protected void create()
{
    super.create();
    setRangeSize(-1);
    executeQuery();
    getAllRowsInRange();
}
```

**Note:**

This step is optional as there may be caches, such as profile cache, that you would want to populate lazily.
For information about how to create a shared application module, see the "Sharing Application Module View Instances" chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

60.7.3 How to Cache Multi-Language Support Data

In addition to the instructions provided in How to Add Data to Cache, you must also perform the following steps to cache multi-language support (MLS) data. These steps are required because the shared application module and view object cache only stripes data by bind parameters. Therefore, you must build your MLS view objects for caching differently than other normal MLS view objects. For example, you must add bind parameters to the MLS cache view objects.

60.7.3.1 Create ADF Business Components Objects for Shared MLS Data

The procedure for creating ADF Business Components objects for shared MLS data varies depending on where the shared data requirement exists. The steps you follow are different for sharing data from the base table, from the translatable, or _TL, table or from both the base and the _TL table.

60.7.3.1.1 How to create objects if only the data from the base table must be shared

1. Create an entity object on top of the base table if you need the change notification feature. This means that your data in cache is refreshed when there is a change in the underlying table. This is required because the database change notification feature doesn't work against database views.

2. Create a view object on top of the _VL entity object if you do not require change notification. Otherwise, create a view object on top of the entity object created from the previous step.

3. Exclude all language dependent attributes from the _VL entity object.

60.7.3.1.2 How to create objects if only the data from the _TL table must be shared

1. Create the entity object on top of the _TL table. This is required by MLS Framework. For more information, see Understanding Multi-Language Support Features.

2. Create a view object on top of the _TL entity object.

3. Create commonly used view criteria, with language being part of the criteria using a bind variable.

Caution:
The language must always be part of any view criteria. This is very important.

60.7.3.1.3 How to create objects if both the data from the base table and the _TL table must be shared

1. Create the entity object on top of the _TL table.

This is required by MLS Framework. For more information, see Understanding Multi-Language Support Features.
2. Create an entity object on top of the base table if you need the change notification feature. This means that your data in cache is refreshed when there is a change in the underlying table. This is required because the database change notification feature does not work against database views.

If you do not need the change notification feature, then you can use the existing _VL entity object, which should have been created already because it is required by MLS Framework.

3. Create a view object to join the entity object created in the previous step (either a _VL entity object or an entity object based on the base table) and the _TL entity object. The view object should have all the language dependent attributes from the _VL entity object excluded, which allows the language dependent attribute to always come from the _TL entity object.

   **Tip:**
   This is important as it allows different users to see data for their language.

4. Create commonly used view criteria, with language being part of the criteria using a bind variable.

   **Caution:**
   The language must always be part of any view criteria. This is very important.

### 60.7.3.2 Create ADF Business Components Objects that Join to MLS tables

The procedure for creating ADF Business Components objects that join to MLS tables varies depending on where the data requirement exists. The steps you follow are different if the data is from the base table, from the translatable, or _TL, table, or from both the base and the _TL table.

#### 60.7.3.2.1 How to create objects if only the data from the base table is required

1. Create an entity object on top of the base table if you need the change notification feature. This means that your data in cache is refreshed when there is a change in the underlying table. This is required because the database change notification feature does not work against database views.

2. Create a view object that joins to the _VL entity object if you do not need the change notification feature, or create one that joins to the entity object created in previous step if change notification feature is required.

3. Exclude all the language dependent attributes from the _VL entity object.

#### 60.7.3.2.2 How to create objects if only data from the _TL table is required

1. Create a view object that joins to the _TL entity object.

2. Create view criteria with language being part of the criteria using a bind variable.

   **Caution:**
   The language must always be part of any view criteria. This is very important.
60.7.3.2.3 How to create objects if data from both the base table and the _TL table is required

1. Create an entity object on top of the base table if you need the change notification feature. This means that your data in cache is refreshed when there is a change in the underlying table. This is required because the database change notification feature does not work against database views.

2. Create a view object that joins to the _VL entity object or the entity object created in the previous step, and the _TL entity object.

3. Exclude all the language dependent attributes from the _VL entity object so that the language dependent attributes always come from the _TL entity object.

   **Tip:**
   This is important as it allows different users to see data for their language.

4. Create commonly used view criteria, with language being part of the criteria using a bind variable.

**Caution:**
The language must always be part of any view criteria. This is very important.

60.7.4 How to Consume Cached Data

The most common approach for accessing the shared data is to create a view accessor. You can also instantiate a shared application module programmatically, if required.

60.7.4.1 Consume Shared Data Using a View Accessor

Follow these steps to consume shared data using a view accessor:

1. Identify the shared application module that contains the shared data.

2. (Optional) Create view accessors on top of a shared view object.

   If the shared view object contains language specific attributes, make sure to include a view criteria that filters by language and bind the language to the current session language when defining your view accessor.

3. (Optional) Build validators on top of the view accessors that you created in Step 2.

   This allows defaulting and derivation, and other business logic to use these view accessors.

4. (Optional) Use the shared view object instead of using entity object as the validation target type for the key exist validator that validates shared data.

   **Tip:**
   If you use entity object target type, it does not use application-level cache.

For information about how to create view accessors, see the Accessing View Instances of the Shared Service section of the "Sharing Application Module View Instances" chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.
60.7.4.2 Create a shared application module programmatically

If you have an existing local application module, use the findOrCreateSharedApplicationModule method to create a shared application module. If you do not have a handle to an existing local application module, then use createRootApplicationModuleHandle from the oracle.jbo.client.Configuration class. Ensure that you release the application module after you are done, for example:

```java
ApplicationModuleHandle handle = Configuration.createRootApplicationModuleHandle("mypkg.AppModule", "AppModuleShared");
ApplicationModule sharedAM = handle.useApplicationModule();
...
Configuration.releaseRootApplicationModuleHandle(handle, false);
```

If you rely upon the database change notification feature to refresh your shared AM cache, then you also need to manually invoke the processChangeNotification method on the shared AM to get the latest data.

For more information, see the "Sharing Application Module View Instances" chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

60.7.5 What Happens at Runtime: When Another Service Accesses the Shared Application Module Cache

During runtime, only one instance of a shared application module is created in the application module pool. If there is an existing application module in the pool, then the existing application module instance is returned when you request a shared application module. For more information, see the "What Happens at Runtime: When Another Service Accesses the Shared Application Module Cache" section in the Developing Fusion Web Applications with Oracle Application Development Framework.

60.8 Profile and Trace Oracle Fusion Applications

To monitor performance in Oracle Fusion Applications you can use the JDeveloper Profiler and capture SQL Trace for Oracle Fusion Applications. For detailed information about monitoring and debugging techniques, see the "Monitoring Oracle Fusion Middleware" chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.

60.8.1 How to Profile Oracle Fusion Applications with JDeveloper Profiler

The JDeveloper Profiler is used to provide information about the CPU, elapsed time, and memory metrics, as well as call counts. It can be very helpful when you are dealing with a performance issue or just trying to understand the performance characteristics of your code.

For more information about JDeveloper Profiler, consult the JDeveloper Help documentation.

Useful profiling modes are:

- **Sample CPU time with Collect Elapsed Time**: Using this mode, you can find the methods using the most CPU. The Collect Elapsed Time option also shows the method taking the most time, including time spent in the database. This mode has low overhead and does not significantly slow down the application.
If you find a method with a high elapsed time but low CPU time and that method includes a database call, this could indicate either a slow query or too many database roundtrips between the database and the middle-tier over a slow network. Look for methods with the highest exclusive CPU (sort on the CPUx field), and use the stack trace to determine where they are called from and if they can be optimized.

- **Memory Profiling**: Using this mode, you can find out how much memory is allocated during the test.
- **Call Count Profiling**: This is part of the CPU profiler and can be used to find out how many times each method is called.

**Caution:**

Call count profiling has very high overhead and therefore, you should increase Oracle JDeveloper starting memory before using it.

To reduce resource consumption, set appropriate filters to include only the classes you are interested in.

### 60.9 Set up a Debug Breakpoint

If you are interested in where a certain method is called, you can set a breakpoint on that method and capture the stack trace. You can do this either interactively or preferably, you can set up a debug breakpoint at the target line and print the stack automatically.

**To set up a debug breakpoint at the target line:**

1. Highlight your breakpoint in the Breakpoints page.
2. Click **Edit** and select the **Actions** tab.
3. Deselect the **Halt Execution** option and select the **Log Breakpoint Occurrence** and the **Stack** option. Selecting the **Stack** option gives you the stack trace, as shown in the following figure.
Each time the breakpoint is hit, the stack is written to the console. To capture this, you must log the console output to a file.

**To log the console output to a file:**

1. Go to **Tools, Preferences** and select the **Environment: Log** category.

2. Select the **Save Logs to File** option and specify the Log directory, as shown in the following figure.

**Figure 60-12 Preferences — Environment: Log**

After running your project, you can find the console logged to a file in the specified directory.
This chapter describes the process of debugging your Oracle Application Development Framework (Oracle ADF) and Oracle SOA Suite applications. It describes how to diagnose and correct errors and how to use the debugging tools.

This chapter includes the following sections:

- Introduction to Debugging and Collecting Diagnostics
- Diagnosing Problems
- Debugging in JDeveloper
- Troubleshooting
- Testing and Troubleshooting

### 61.1 Introduction to Debugging Oracle ADF and Oracle SOA Suite

Debugging Oracle Application Development Framework (Oracle ADF) is a process of collecting and isolating factors that contribute to problems that occur when the web page components interact with the ADF Model layer.

You can use diagnostics tools for collecting contextual information for isolating the problem. One of the most useful diagnostic tools is the ADF Logger. You use this logging mechanism in JDeveloper to capture runtime traces messages. With Oracle ADF logging enabled, JDeveloper displays the application trace in the Message Log window. The trace includes runtime messages that may help you to quickly identify the origin of an application error. Another useful diagnostic tool is SQL trace, which enables tracing of the current database session and logs all SQL statements to a server-side trace file.

Once you have gathered the diagnostic information, you can use the JDeveloper debugging tools to investigate where your application failure occurs. These include the JDeveloper Debugger, which is a comprehensive debugger to assess and repair your code, and the ADF Declarative Debugger for declaratively setting breakpoints on ADF task flow activities, page definition executables, method, action, and value bindings, and Oracle ADF lifecycle phases.

Oracle SOA Suite provides a complete set of service infrastructure components for designing, deploying, and managing composite applications. Test cases enable you to simulate the interaction between a SOA composite application and its web service partners before deployment in a production environment. This helps to ensure that a process interacts with web service partners as expected by the time it is ready for deployment to a production environment.
61.2 Collecting Diagnostics

Collecting diagnostics information helps you to obtain more contextual information for isolating the problem.

61.2.1 How to Collect Diagnostics in the Integrated WebLogic Server Environment

In the Integrated WebLogic Server environment, you can maximize the availability of diagnostics information by:

- Enabling diagnostic logging in the development environment
- Enabling database tracing

61.2.1.1 Enabling Diagnostic Logging in the Development Environment

You can also enable logging in the development environment by setting the Java system property named `jbo.debugoutput` to the value `console`.

**Related Links**

The following document provides more information about the topic discussed in this section:

- For more information, see the "How to Turn On Diagnostic Logging for Non-Oracle ADF Loggers" section in *Developing Fusion Web Applications with Oracle Application Development Framework*.

61.2.1.2 Enabling Database Tracing in Integrated WebLogic Server Instances

Database tracing can be a very useful way of verifying whether the queries executed by ADF Business Components are actually returning any data. The ADF Business Components tracing output (`-Djbo.debugoutput=console`) lists the query and the bind variable, but it is not always clear how many rows are fetched or how the fetching takes place, that is, in batches or one row at a time. If you are investigating a suspected bug or performance issue in ADF Business Components, it is always good to have the database trace to help understand the problem.

Database tracing is usually used for performance tuning. You should know how to generate a SQL trace, find the query you are interested in, check the bind variables, and tell how many rows were returned. It is usually the quickest way of telling whether missing data in your application is a middle tier application bug, or missing data in the database. It is also useful when you are investigating ORA errors being returned from the database.

You should consider the following with database tracing:

- When you test using Integrated WebLogic Server with Oracle WebLogic Server data sources, the database connection is obtained from `fusion_apps_wls.properties` and not from `connections.xml`. However, when you use the application module tester to run an application module, the connection details from `connections.xml` are used, even when you set your application module configuration to use an Oracle WebLogic Server data source. Since the application module tester does not use Oracle WebLogic Server, it builds a database connection from the information in `connections.xml`.

- If you change the database details in `fusion_apps_wls.properties`, the updates are not picked up until the Oracle WebLogic Server domain is re-created.
Shutting down the Integrated WebLogic Server instance is not enough. The domain for the Integrated WebLogic Server is automatically created when you launch the Integrated WebLogic Server instance for the first time in a new Oracle Fusion Applications ADE view, or re-created the next time you run the Integrated WebLogic Server instance after deleting the domain directory. If in doubt, you can view the ApplicationDBDS JDBC data source in the Oracle WebLogic Server Administration Console to see what database it is pointing to. You can change the database in the Oracle WebLogic Server Administration Console to point to the new database, but the next time the domain is re-created it will be set to the connection defined in fusion_apps_wls.properties.

- When you try to examine data from a SQL*Plus session for views based on translated data, check that `select userenv('lang') from dual; returns US`. If it does not, change the language with:
  ```sql
  alter session set nls_language='American';
  ```
  If you do not do this, some of the translated view will not return any data. For example, if you are based in the United Kingdom, `select userenv('lang')` may return `GB` by default, which does not match any of the data in the TL tables. This is particularly important in the development environment where only the US messages are available.

- There may be other security restrictions that prevent you from viewing certain data from a SQL*Plus session, such as data that is protected by virtual private database (VPD) or data that requires you to initialize your `userenv`.

- You may receive ORA-600 errors, which typically manifest themselves as ORA-3114 or ORA-3113 on the client and indicate that the database session has terminated abnormally. In this case, a trace file is always created automatically, and the database alert log is updated with a record of the failure and the name of the trace file. You do not need to enable SQL trace in this case because the trace file is always created and should include a full dump, which will help database Support and Development diagnose the cause. Both the alert log and the trace file will be in the database User Directory.

You can also enable Oracle ADF tracing. For information, see the "Use SQL Tracing to Identify Ill-Performing Queries" section in Developing Fusion Web Applications with Oracle Application Development Framework.

### 61.2.2 How to Collect Diagnostics in the Standalone WebLogic Server Environment

In the standalone WebLogic Server environment, you can maximize the availability of diagnostics information by:

- Enabling diagnostic logging
- Adding debug messages to your code
- Enabling database tracing

#### 61.2.2.1 Enable Diagnostic Logging in the Provisioned Environment

Use logging profile options to enable diagnostic logging in the provisioned environment so you can view your log statements in the directory configured against your apps-handler.

```
${domain.home}/servers/${weblogic.Name}/logs/apps/${weblogic.Name}-diagnostic-log
```
To enable diagnostic logging in the provisioned environment:

1. Get the user GUID for which you want to enable JBO logging. You can obtain the user GUID from the `WLS_Users_and_Groups.ldift` file by searching for "uid=". The string for `orclguid` is the user GUID.

2. Add the following logging profile options for the required user:
   - AFLOG_ENABLED USER Y to enable logging for the user.
   - AFLOG_LEVEL USER 300 to set the level to the lowest severity (FINEST) for the user.
   - AFLOG_MODULE USER % to enable logging for all modules for the user.

   For information on setting the profile options for logging, see Defining Profiles.

3. Make sure your `logging.xml` has the `oracle.apps` logger configured against a handler. In your standalone server, `logging.xml` is located in the standalone domain at `<.....> / domains/fusion_domain/config/fmwconfig/servers/AdminServer/logging.xml`.

   The following logger should exist in `logging.xml`:
   ```xml
   <logger name='oracle.apps' level='ALL'
       useParentHandlers='false'>
       <handler name='apps-handler'/>
   </logger>
   ```

   The handler `apps-handler` should also exist in the handlers section.

### 61.2.2.2 Adding Debug Messages to Your Code

To help you investigate problems in the standalone WebLogic Server instance, you can add debug messages to your code so that messages, such as `System.out.println()`, which are normally displayed on the Java console, are written to the server log file (`AdminServer.log`) in the default domain.

Since the log file may get rather large, particularly if more than one user is using the environment, you may want to prefix your messages with an identifier that you can easily find through a search.

Log in to the Environment Management System (EMS) WebLogic Server host using the Applmgr username and password. The default password is the username in uppercase letters, but your family environment owner may have changed it, so check with them if necessary. The server log should be in the Applmgr home directory. To locate other log files, use the `find . -name "*.log"` command.

### 61.2.2.3 Enabling Database Tracing in Standalone WebLogic Server Instances

Enabling database tracing allows the standalone WebLogic Server instance to write all your actions to an associated trace file. When an internal error is detected by a process, it dumps information about the error to its corresponding trace file.

#### 61.2.2.3.1 Enable Database Tracing

Enable database tracing before you start your test flow.
To enable database tracing:

1. Sign in to the Oracle Fusion application with a user account that is provisioned with the necessary role, such as the predefined Application Implementation Consultant role. Contact your security administrator for details.

2. From the Help menu in the work area of the Oracle Fusion application, choose Troubleshooting and select Troubleshooting Options.

3. In the Options dialog, select the Database trace checkbox.

   This option enables SQL trace for all database connections used by the current user session.

   You can also select options to enable the SQL trace option to capture bind variables and wait events.

61.2.2.3.2 Locate Your Trace File

The trace file can be found in the USER_DUMP_DEST directory specified by the user_dump_dest init.ora parameter, which is usually ORACLE_HOME/log/diag/rdbms/sid/sidtrace. The trace filename includes the FND session ID appended to the end, for example, mysid ora 4473 881497BF7770BEEE040E40A0D807BB1.trc. You must identify the session ID to locate your trace file.

Note:

From SQL*Plus, you can execute SQL> show parameter user to show user_dump_dest. An operation system login is required to access this directory.

To identify the session ID in Mozilla Firefox:

1. From the Tools menu in Firefox, choose Options, then select the Privacy panel.

2. From the Firefox will list, select Use custom settings for history, then click Show Cookies.

3. In the Search field, enter Oracle, then look for a cookie that contains FND_SESSION in the name.

4. Inspect the value of the cookie, for example, DEFAULT_PILLAR: BsdhOZScx9NeAA...:12490558567373.

   Note the middle portion (using : as separator), for example, BsdhOZScx9NeAA...

5. Open a SQL*Plus session (log in as fusion user) and locate your Applications Core session ID by executing:

   select session_id from fnd_sessions where session_cookie = value from Step 4

   For example,

   select session_id from fnd_sessions where session_cookie = 'BsdhOZScx9NeAA...'

   The value returned if your session ID, which you can use to locate your trace file.
61.3 Diagnosing Problems

In addition to reviewing the diagnostics information, you can perform various tasks to diagnose problems in your server environment.

61.3.1 How to Diagnose Problems in the Integrated WebLogic Server Environment

Perform the following tasks to diagnose problems in the Integrated WebLogic Server environment:

61.3.1.1 Test the JDBC Data Source Connections

While you are diagnosing problems in the Integrated WebLogic Server environment, you may want to verify that the JDBC data source connections are pointing to the correct database connection string.

To test the connections:

1. From the JDeveloper menu, select Run, then Start Server Instance.
2. Select Application Server Navigator from the View menu.
3. Expand the Application Servers folder, then right-click IntegratedWebLogicServer, and select Launch WebLogic Console.
4. Log in to the Integrated WebLogic Server console using the username and password weblogic and weblogic1.
5. In the Domain Structure tree of the Oracle WebLogic Server Administration Console, navigate to Services, then JDBC, then Data Sources.
6. From the list of configured data sources, select each data source and click the Test button to make sure that the specified database can be reached with the connection information.

61.3.1.2 Viewing the Application Module Pooling Statistics

When running on Integrated WebLogic Server, you can view the application module pooling statistics to verify that the domain is properly configured for Oracle ADF.

To view the application module pooling statistics:

2. Log in as an administrator.
3. Click the ampool metric.
4. Scroll your browser to the right to view all the statistics.

61.3.1.3 Sanity Checking Your EAR File in the Integrated WebLogic Server Environment

Sanity checking your EAR file helps to diagnose problems in the Integrated WebLogic Server environment. Download the EAR file and open it using a decompression utility. You can then drill down into the WAR file and individual Oracle ADF libraries.

While sanity checking your EAR file, verify the following:
• The UI libraries are in the `WEB-INF/lib` directory of the WAR file.
• The Model libraries are in the `APP-INF/lib` directory of the EAR file.
• The service middle tier JAR and the service WAR files are directly under the EAR file.
• The service common JAR file is directly under the EAR file or under the `APP-INF/lib` directory, depending on how it was set up.
• The individual Oracle ADF libraries only contain components from the project that was deployed to create the Oracle ADF library, and do not contain any components from referenced projects. (This could happen if you have "build output" dependencies.) If components from other projects were included, it should be obvious from the package, since every project has a unique default package.
• All standalone components (that is, those not in an Oracle ADF library) in the WAR file belong to the UI project that was deployed (such as SuperWeb for Oracle Fusion Applications). There should therefore be no standalone model components in the WAR file.
• None of the Model and Service Oracle ADF libraries have a `public_html` directory.
• No technology JAR files are included in the EAR and WAR files. Tech stack JAR files added to the WAR or EAR file take precedence over the ones in the shared libraries, but the shared libraries contain the correct versions. The technology JAR files in your EAR or WAR file may not be the correct versions.

61.3.2 How to Diagnose Problems in the Standalone WebLogic Server Environment

Perform the following tasks to diagnose problems in the standalone WebLogic Server environment:

• Sanity check your enterprise archive (EAR) file
• Examine the Oracle WebLogic Server classloaders

61.3.2.1 Sanity Checking Your EAR File in the Standalone WebLogic Server Environment

The procedure for sanity checking your EAR file in the standalone WebLogic Server environment is the same as the procedure for Integrated WebLogic Server. For information, see Sanity Checking Your EAR File in the Integrated WebLogic Server Environment.

61.3.2.2 Examining the Oracle WebLogic Server Classloaders

Using a tool called CATX (Classloader Analysis Tool), you can diagnose problems in the standalone WebLogic Server environment by examining the JAR files and loaded classes when your application is running from Oracle WebLogic Server.

CATX is a web application that is deployed by default to every Oracle Fusion Applications Oracle WebLogic Server domain. To launch CATX, run `http://host:port/catx/`. 
With CATX, you can identify a duplicate class by determining from which JAR file a class file was loaded. You can also identify any other locations where the class is duplicated in the J2EE application classpath or web application classpath.

As an alternative to CATX, you can add the following code to your custom classes to display in the console the JAR file from which the class was loaded:

```
File moduleFile = new File
(YOURCLASSHERE.class.getProtectionDomain().getCodeSource().getLocation().
toURI());
System.out.println("Jar file name = " + moduleFile.fullPath());
```

This method may be useful if you are actively debugging code from JDeveloper or using the AM Tester.

### 61.4 Debug in JDeveloper

The following debugging tools are available for debugging in JDeveloper:

- JDeveloper Debugger.
  
  For information, see the JDeveloper online help.

- ADF Declarative Debugger.

**Related Links**

The following document provides more information about the topic discussed in this section:

- For information, see the "Using the ADF Declarative Debugger" section in *Developing Fusion Web Applications with Oracle Application Development Framework*.

### 61.4.1 How to Debug an Application Remotely

Use JDeveloper to perform remote debugging for your application.

Oracle WebLogic Server logs and Fusion Middleware ODL (Oracle Diagnostic Logging) logs are located in the following directory:

```
 JDEV_USER_HOME/system11.1.1.32.52.37/DefaultDomain/servers/DefaultServer/logs/
```

**To perform remote debugging of your application:**

1. `setenv debugFlag true`
2. Start Oracle WebLogic Server.
3. Using JDeveloper, go to Application Navigator and select your ViewController project.
4. Select Run, Choose Active Run Configuration, Manage Run Configurations.
5. Select New and enter a name for the new profile. For example, Remote Debug.
6. Click OK.
7. Edit the new Remote Debug profile by entering the following information:
   - a. Add a default Run Target. For example:
b. Select the remote Debugging and Profiling option.

c. Go to Debugger, Remote. Enter the following information:
   Protocol: Attach to (Java Platform Debugger Architecture (JPDA))
   Host: Enter the host where Oracle WebLogic Server is running.
   Port: Enter the Java Debugger Wire Protocol (JDWP) port number.

   The Java Virtual Machine (JVM) is listening for JDPA requests. The default port is 8453. Use the `setenv DEBUG_PORT new port number` command to change the port number.

**Note:**
Alternatively, you can create a profile by selecting your ViewController project, right-click and select Project Properties, Run/Debug/Profile.

8. On the toolbar, click the Bug icon. Select the new Remote Debug profile. A dialog appears to confirm attaching to the Administration Server.

   The debugging log file will contain an entry similar to the following:

   Debugger attempting to connect to remote process at ab6052cdef.us.example.com
   8453. Debugger connected to remote process at ab6052cdef.us.example.com
   8453. Processing 10008 classes that have already been prepared...Finished processing prepared classes. Debugger process virtual machine is Java HotSpot(TM) Server VM.

   Set your break points and initiate your program. When you have finished, click the red Stop button, and a dialog will appear asking if you want to Detach, Terminate, or Cancel. Detach detaches from the remote Oracle WebLogic Server, and Terminate terminates the remote Oracle WebLogic Server.

   **Tip:**
   Only one developer can debug at a given time on a specific port.

### 61.5 Troubleshooting Oracle ADF

This section describes common problems that you might encounter when using Oracle ADF with Oracle Fusion Applications and explains how to solve them.

#### 61.5.1 Problems and Solutions

The following are common problems you may encounter and solutions that solve them:

- "Too many files" Error Occurs on Local Linux Servers
- Compilation Error Occurs
- "No def found" or "No class def found" Exception Occurs
- Breakpoints Are Not Functioning Correctly
- Empty List in the Data Controls Panel
• Runtime Error Related to DataBindings.cpx File
• "Application module not found" Errors Related to DataBindings.cpx File
• Hot Reloading Does Not Work
• Missing ADF Component at Runtime in
• Odd ADF Component Errors
• is Not Responding
• Missing Base Class
• Unavailable FND Components
• JavaServer Pages Compilation Errors
• ApplicationDB Errors While Running the Integrated WebLogic Server
• Metadata Services Runtime Exception
• Application Cannot Fetch Data from Database
• "The task cannot be processed further" Message Appears
• TimedOut Exception Occurs

61.5.1.1 "Too many files" Error Occurs on Local Linux Servers
You receive an error that indicates that too many files are open.

Problem
The open file limit on local Linux servers has been exceeded.

Solution
Increase the open file limit.
For information, see Increasing Open File Limit on Linux Machines.

61.5.1.2 Compilation Error Occurs
You encounter a compilation error.

Problem
A reference to an Oracle ADF business component or Java class in an Oracle ADF library cannot be resolved, such that it does not exist or is incompatible with the existing reference.

Solution
All references to components contained in Oracle ADF libraries are resolved when the workspace is loaded in JDeveloper. Refresh the library in one of the following ways:

• Close the workspace and re-open it to refresh the references to the Oracle ADF libraries. (Closing and restarting JDeveloper with a workspace open does not refresh the references to the Oracle ADF libraries.)

• If you have a specific project selected in the JDeveloper navigator pane, select View, then Refresh ADF Library Dependencies for ....jpr to refresh the references to the Oracle ADF libraries.
When you make any changes to the components in a project, where the components are being referenced as an Oracle ADF library by your user interface (UI) project, you must redeploy the Oracle ADF library and refresh the Oracle ADF library dependencies for your UI project. The same applies to one model project referencing from another model project. If you are developing or debugging code in a model project while running the referencing UI project to test it, it may be easier to add the model project as a build output dependency, so you do not need to go through the cycle of redeploying the Oracle ADF library or refreshing the Oracle ADF library references each time you make a change.

61.5.1.3 "No def found" or "No class def found" Exception Occurs

Either a *No Def Found* or *No Class Def Found* runtime exception occurs.

**Problem**

Lower level dependency changes were made outside of the design time session.

**Solution**

Refresh the library in one of the following ways:

- Close the workspace and re-open it to refresh the references to the Oracle ADF libraries. (Closing and restarting JDeveloper with a workspace open does not refresh the references to the Oracle ADF libraries.)
- If you have a specific project selected in the JDeveloper navigator pane, select *View*, then *Refresh ADF Library Dependencies for ....jpr* to refresh the references to the Oracle ADF libraries.

**Note:**

When you make any changes to the components in a project, where the components are being referenced as an Oracle ADF library by your user interface (UI) project, you must redeploy the Oracle ADF library and refresh the Oracle ADF library dependencies for your UI project. The same applies to one model project referencing from another model project. If you are developing or debugging code in a model project while running the referencing UI project to test it, it may be easier to add the model project as a build output dependency, so you do not need to go through the cycle of redeploying the Oracle ADF library or refreshing the Oracle ADF library references each time you make a change.

61.5.1.4 Breakpoints Are Not Functioning Correctly

Execution stops before or after the line with the breakpoint, depending on whether you have added or removed lines of code from the source.

**Problem**

If you consume the components from another project in the same workspace and run in debug mode, you can open the source Java classes from the referenced project in the JDeveloper edit window and set breakpoints. However, if you consume the components at runtime through an Oracle ADF library, the compiled objects from the
Oracle ADF library may not be synchronized with the source if you made changes since you last deployed the Oracle ADF library. If you are using a build output dependency, then you are not affected by this issue.

**Solution**

Redeploy the Oracle ADF library to synchronize the source code.

61.5.1.5 Empty List in the Data Controls Panel

The Applications Core wizards display an empty list of data controls after you make changes to an application model and add additional view object instances or additional view criteria to the view objects.

**Problem**

The Data Controls panel in JDeveloper was not opened in a new view or refreshed following changes made to the data bound Applications Core component.

**Solution**

If using the Applications Core wizards to create an applications panel, applications table, or other data bound Applications Core component, you must open the Data Controls panel in JDeveloper at least once in a new view, or at least once after deleting the system directory, before launching the Applications Core wizards. If necessary you should refresh from the Data Controls panel before launching the Applications Core component wizards.

61.5.1.6 Runtime Error Related to DataBindings.cpx File

A runtime error occurs while the runtime model is being located using the information in the DataBindings.cpx file.

**Problem**

You have more than one UI project (for example, task flows referenced from an Oracle ADF library) when the DataBindings.cpx files are merged at runtime.

**Solution**

Make sure that each instance of DataBindings.cpx resides in its own package.

61.5.1.7 "Application module not found" Errors Related to DataBindings.cpx File

An Application module not found error occurs.

**Problem**

The DataBindings.cpx file was moved because the default package was set incorrectly.

**Solution**

Make sure that the package defined in the Application tag of DataBindings.cpx matches the current package location.

61.5.1.8 Oracle WebLogic Server Hot Reloading Does Not Work

You get unexpected behavior, blank components, or unexpected exceptions when you hot reload in either of the following ways:

- In the Message Log window, select the Running Integrated WebLogic Server tab and click the URL that launched your page in Integrated WebLogic Server.
• Remove the Oracle ADF state related information from the URL displayed in the browser (for example, ?_adf.ctrl-state=ku8gus1cz_4) and reload the page.

**Problem**

Changes were made to the page binding definition file (PageDef), the resource (XLF) files, or the Oracle ADF libraries.

**Solution**

Generally if the change you make is contained within a page or page fragment of the current project, you do not need to redeploy your application. However, if changes are made to the page binding definition file (PageDef), the resource (XLF) files, or the Oracle ADF libraries then you must redeploy your application.

**61.5.1.9 Missing ADF Component at Runtime in Oracle WebLogic Server**

You discover a missing ADF component at runtime in Oracle WebLogic Server.

**Problem**

There is a second version of the ADF component erroneously included when the Oracle ADF libraries and WAR or EAR files were built.

**Solution**

Check the EAR file to make sure that the missing component is actually present in the expected location.

**61.5.1.10 Odd ADF Component Errors**

You receive odd errors related to an ADF component that suggests a recent change was not picked up.

**Problem**

A second copy of that component is incorrectly referenced in another Oracle ADF library via a build output reference.

**Solution**

Perform one of the following tasks to resolve the problem:

• Refresh the library in one of the following ways:
  
  – Close the workspace and re-open it to refresh the references to the Oracle ADF libraries. (Closing and restarting JDeveloper with a workspace open does not refresh the references to the Oracle ADF libraries.)
  
  – If you have a specific project selected in the JDeveloper navigator pane, select View, then Refresh ADF Library Dependencies for ....jpr to refresh the references to the Oracle ADF libraries.
Note:
When you make any changes to the components in a project, where the components are being referenced as an Oracle ADF library by your user interface (UI) project, you must redeploy the Oracle ADF library and refresh the Oracle ADF library dependencies for your UI project. The same applies to one model project referencing from another model project. If you are developing or debugging code in a model project while running the referencing UI project to test it, it may be easier to add the model project as a build output dependency, so you do not need to go through the cycle of redeploying the Oracle ADF library or refreshing the Oracle ADF library references each time you make a change.

- Force recompilation by right-clicking the JSP and selecting Build, then Clean All, then rebuild and redeploy.

61.5.1.11 Oracle WebLogic Server is Not Responding
The Oracle WebLogic Server instance may seem unresponsive.

Problem
The Oracle WebLogic Server Java process may be CPU intensive.

Solution
Use `kill -3 pid` to write a Java thread dump to the administration console for Integrated WebLogic Server or to the server log file for standalone WebLogic Server.

`kill -3 pid` is the same as `kill -QUIT pid`, which sends SIGQUIT to the process.

The Java process implements a signal.

If you are using Jrockit Java, use this command to print threads:

```
jrcmd <pid> print_threads [nativestack=true]
```

61.5.1.12 Missing Base Class
The base class is missing when you test the Model project in the Oracle Business Component Browser.

Problem
The Model project is missing the Applications Core library.

Solution
Add the Applications Core library to the Model project.

61.5.1.13 Unavailable FND Components
The FND components are not available when you add components to your page or page fragment.

Problem
The Applications Core (ViewController) tag library is missing.

Solution
Add the Applications Core (ViewController) library to your project to automatically add the Applications Core (ViewController) tag library. You may need to close the
Project Properties dialog, save the changes, and reopen the Project Properties dialog before you see all the dependent changes made when adding the Applications Core library.

61.5.1.14 JavaServer Pages Compilation Errors

You get JavaServer Pages (JSP) compilation errors or other JSP errors.

Problem

The page or page fragment is invalid. In the source editor, you can see that there are errors in the page, often because of malformed XML (for example, missing or mismatched XML tags) or some other error reported by the design time audits. This can occur if you cut and paste directly into the XML source. JDeveloper allows you to run the page even though it is invalid.

Solution

In the Preferences option of the Tools menu, you can set an audit to run during compilation. If there are failures, it will prevent the run. However, the audit only executes during compilation. The first time you try to run, it may need to compile and the audit kicks in and it fails, which causes the run to stop. The second time you try to run, everything that needs to compile may have already successfully compiled. In this case, there is no compilation, so there is no audit.

61.5.1.15 ApplicationDB Errors While Running the Integrated WebLogic Server

You get errors related to ApplicationDB when you are running the Integrated WebLogic Server.

Problem

The settings for ApplicationDB are not configured correctly.

Solution

Check the settings for the ApplicationDB in the Oracle WebLogic Server Administration Console.

61.5.1.16 Metadata Services Runtime Exception

You get the following Metadata Services (MDS) runtime exception:

```
oracle.mds.exception.MDSRuntimeException: MDS-02401: The operation ModifyAttribute on the showDetailItem node is not allowed.
MDS-02404: The subelement RAshowdetail2 in the MDX document does not allow customization of /oracle/apps/fnd/applcore/patterns/uishell/RegionalArea.jsff#RAshowdetail2.
```

Problem

The CHANGE_PERSISTENCE parameter value is incorrect.

Solution

Locate the following context parameter in `web.xml`:

```xml
<context-param>
  <description>
    This parameter turns on the session change persistence.
  </description>
  <param-name>org.apache.myfaces.trinidad.CHANGE_PERSISTENCE</param-name>
```
Change the value of CHANGE_PERSISTENCE to oracle.adf.view.page.editor.change.ComposerChangeManager.

### 61.5.1.17 Application Cannot Fetch Data from Oracle Fusion Applications Database

Your application is unable to fetch data from the Oracle Fusion Applications database.

**Problem**

The fusion_apps_wls.properties file does not contain the correct connection strings for the application’s data source.

**Solution**

Run the Configure Fusion Domain Wizard to create or update the fusion_apps_wls.properties file with the correct connection information. For instructions on using the wizard, see Chapter 2, "Setting Up Your Development Environment."

### 61.5.1.18 "The task cannot be processed further" Message Appears

You get the following message:

Deployer: 149140 The task cannot be processed further until the current edit session is activated.

**Problem**

You modified the configuration of the server and did not activate the changes.

**Solution**

Go to the Administration Console (http://localhost:7101/console). Check the upper left hand corner regarding messages about changes not being activated.

### 61.5.1.19 TimedOut Exception Occurs

You get the following exception:

weblogic.transaction.internal.TimedOutException

Service logic is taking longer than the default 300 seconds defined for Java Transaction API (JTA). Services may have heavy validation which will take more time to create row.

**Solution**

Set the JTA timeout condition to more than 300.

2. Log in using your username and password.
3. Go to Domain Structure, choose Service.
4. Go to Services and select JTA.
5. Increase the value of the Timeout Seconds property based on the expected completion time of the longest transaction. The default value is 300.
61.6 Testing and Troubleshooting Oracle SOA Suite

For more information about testing and troubleshooting SOA composite applications, see the "Testing and Troubleshooting" section in Developing SOA Applications with Oracle SOA Suite.

You can also automate the SOA composite applications testing. For information, see the "Automating Testing of SOA Composite Applications" section in Developing SOA Applications with Oracle SOA Suite.
This chapter provides guidelines and best practices for designing and securing Oracle Application Development Framework (Oracle ADF) view objects and other supporting business component objects for use by Oracle Business Intelligence Applications.

This chapter includes the following sections:

- Introduction to View Objects for Oracle Business Intelligence Applications
- Understanding General Design Guidelines
- Understanding Design Patterns
- About Designing and Securing Fact View Objects
- About Designing and Securing Dimension View Objects
- About Designing Date Dimensions
- About Designing Lookups as Dimensions
- About Designing and Securing Tree Data
- About Supporting Flexfields for Oracle Business Intelligence Applications
- About Supporting SetID
- About Supporting Multi-Currency

### 62.1 Introduction to View Objects for Oracle Business Intelligence Applications

The view objects that are designed and created for Oracle Business Intelligence Applications (Oracle BI Applications) are shared between Oracle Transactional Business Intelligence and Oracle BI Applications.

The Oracle BI Applications warehouse is populated from Oracle Fusion Applications databases using an ETL (extract, transform, and load) process. The ETL process uses the tool to source data from the source system (Oracle Fusion Applications) to the target Oracle BI Application stable. The extract from the source system is done using the Oracle Application Development Framework (Oracle ADF) view objects.

*Figure 62-1* illustrates the Oracle Business Intelligence architecture.
Oracle BI Enterprise Edition (Oracle BI EE) needs to efficiently access data from two or more master/detail-linked view objects to aggregate, present, or report on that combined data set. An essential requirement is to efficiently retrieve the multiple-levels of related information as a single, flattened query result, to perform subsequent aggregation or transformation on it. Oracle ADF Composite View Object API allows the caller to create a new view object at runtime that composes the hierarchical results from two or more existing view-linked view objects into a single, flattened query retrieving the same effective set of data.

From a performance perspective, such queries would need to be performed on low-level data in Oracle BI EE, since the Oracle ADF layer does not directly support aggregation. This would generally slow query performance down. Also, going through additional servers (that is, JavaHost and Oracle ADF) in the network would also be slower than directly querying the database. Therefore, the SQL Bypass feature has been introduced to directly query the database and push aggregations and other transformations down to the database server, where possible, thereby reducing the amount of data streamed and worked on by Oracle BI EE.

The SQL Bypass functionality in Oracle BI EE utilizes the Composite View Object API feature to construct and return a flattened SQL Bypass query that incorporates all of the required columns, filters, and joins required by the Oracle Business Intelligence query. Oracle BI EE then executes this query directly against the database.

### 62.2 Understanding General Design Guidelines

When designing view objects for Oracle Business Intelligence Applications, you should use the following guidelines with regards to entity objects, associations, view objects, view links, and view criteria.
62.2.1 About Entity Object Guidelines

An entity object represents a row in a database table and simplifies modifying its data by handling all data manipulation language (DML) operations for you. It can encapsulate business logic for the row to ensure that your business rules are consistently enforced. Entity objects are required for all Oracle Business Intelligence view objects to support SQL pruning of declarative view objects and to leverage many Fusion specific features. For more information see "Creating a Business Domain Layer Using Entity Objects" in Developing Fusion Web Applications with Oracle Application Development Framework.

All attributes from the physical table (except special, highly sensitive attributes) should be exposed on the entity objects.

62.2.2 About Association Guidelines

An association reflects relationships between entity objects and can be by either reference or composition. All view objects composed of multiple entity objects are flattened using entity object associations.

For more information about associations, see "Creating Entity Objects and Associations" in Developing Fusion Web Applications with Oracle Application Development Framework. For more information about flattening, see Understanding Flattened View Objects.

62.2.3 About View Object Guidelines

A view object represents a SQL query. You use the full power of the familiar SQL language to join, filter, sort, and aggregate data into exactly the shape required by the end-user task. For more information, see "Defining SQL Queries Using View Objects" in Developing Fusion Web Applications with Oracle Application Development Framework.

This section includes some technical requirements, how to use declarative SQL mode, and guidelines regarding view object attributes and outer joins.

62.2.3.1 Understanding the Technical Requirements

The following are the technical requirements driven by use of the Composite View Object API, SQL Bypass, and SQL Pruning.

**Composite View Object API**

- Use view links to establish relationships between view objects.
- View links must not contain custom SQL functions such as `TRUNC` and `BETWEEN`.
- Use the `BI_JOINTYPE` custom view link property to define outer joins on view links.
- There is no support for Java or Groovy calculated attributes.
- Programmatic view objects, transient view objects, and transient attributes are not supported.

**SQL Bypass**

- Full SQL can be obtained at runtime using `vo.getQuery()`.
- There is no support for transient attributes.
• View objects must not contain bind parameters.
• There is no support for Java logic or Java calculated attributes.
• Do not apply data security view criteria programmatically.
• If you are using Multi-Organization Access Control (MOAC) you must not enable MOAC for the view objects for Oracle Business Intelligence Applications. You should use the underlying Fusion Data Security instead.

For more information, see "About Specifying a SQL Bypass Database" in Metadata Repository Builder’s Guide for Oracle Business Intelligence Enterprise Edition.

SQL Pruning

• You should create your view objects in Declarative SQL Mode.
  For more information about Declarative SQL Mode, see "Working with View Objects in Declarative SQL Mode" in Developing Fusion Web Applications with Oracle Application Development Framework.
• You should set primary entity usage to identify the fact and dimension grain because primary entity usage cannot be pruned.
• You must set the Selected in Query property for non-primary key and unsecured attributes to false.
• You should limit view criteria on non-primary entity derived attributes because attributes used in applied view criteria cannot be pruned.
• You should limit order by clauses on non-primary entity derived attributes because attributes used in applied order by clauses cannot be pruned.

62.2.3.2 About View Object Attributes Guidelines

• As a general rule, you should include all attributes from the underlying primary and reference entity objects in your view objects.
  Flex attributes are an exception from this rule. These attributes are not required because they are exposed using the Flex Extender utility.
• You should only include name and description attributes from the reference entity objects that are included to only resolve ID and Code columns.
• You should include Standard Who Columns from all participating entity objects on your view objects for Oracle Business Intelligence Applications. This is to support Oracle BI Applications’s Change Data Capture requirements.
  Exceptions include entity objects that are only included to resolve ID and Codes into meaningful descriptions. For example, entity objects included to only resolve Business Transactional Intelligence-only attributes into a view object using entity object associations.
  Table 62-1 shows the Standard Who Columns.

Table 62-1 Standard Who Columns

<table>
<thead>
<tr>
<th>Standard Who Columns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATED_BY</td>
<td>The user who created the row.</td>
</tr>
</tbody>
</table>
### Table 62-1  (Cont.) Standard Who Columns

<table>
<thead>
<tr>
<th>Standard Who Columns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATION_DATE</td>
<td>The date and time the row was created.</td>
</tr>
<tr>
<td>LAST_UPDATED_BY</td>
<td>The user who last updated the row.</td>
</tr>
<tr>
<td>LAST_UPDATE_DATE</td>
<td>The date and time the row was last updated.</td>
</tr>
<tr>
<td>LAST_UPDATE_LOGIN</td>
<td>The session login associated to the user who last updated the row.</td>
</tr>
</tbody>
</table>

- You should set the **Selected in Query** property to be `false` on all non-primary key view object attributes.
- You should resolve duplicate attribute names on view objects, which are made up of multiple entities, by using an attribute prefix. Use an alias property as both the table alias and column alias in the SQL as well as the view object attribute prefix. For example:
  - The `POLinesVO` includes both the `HeaderEO` and the `LinesEO`.
  - The `LinesEO` is specified as the primary entity on `POLinesVO`. The `HeaderEO` is specified as a reference entity.
  - This view object includes `HeaderId` attributes from both `HeaderEO` and `LinesEO`.
  - To avoid duplication of attributes across `Header` and `Lines` entities, an entity object alias is specified. For example, `Header` and `Lines` for `HeaderEO` and `LinesEO` respectively.
  - The `POLinesVO` is then created using `Header` as the prefix for all `Header` attributes, and `Lines` as the prefix for all `Lines` attributes. For example, `HeaderHeaderId` and `LinesHeaderId`; `HeaderBusinessUnitId` and `LinesBusinessUnitId`.

- Use the following guidelines to resolve view object foreign keys:
  - If the foreign key is a dimension, Oracle Business Intelligence requests a dimension view object and a view link to the dimension view object.
  - If the foreign key is a warehouse domain, Oracle BI Applications requests a view object for ETL. No view link is requested. Oracle BI EE lookup functionality is used to resolve foreign keys.
  - If the foreign key is neither a dimension nor a warehouse domain, you should resolve the foreign key using entity object associations. For MLS-enabled entities, `ID` and `Code` attributes should be resolved using `_VL` views.

#### 62.2.3.3 About Outer Joins

An outer join is generally required when creating a view object based on multiple entity objects, so as to handle situations when not all of the reference entities’ values are present. The specific outer join type (left, right, or full) used should be determined based on the expected data relationships between the primary and reference entities. Note, however, that in some cases, security considerations will require an inner join,
instead. (For an example, see About Designing Fact View Objects.) If a join is required to resolve an ID or Code attribute, use a _VL view instead.

62.2.4 Understanding View Links Guidelines

View links are required to flatten view objects using the Composite View Object API. To define outer joins on view links, you must add the BI_JOINTYPE custom property on the view link definition. Valid values for this custom property include:

- LEFTOUTER
- RIGHTOUTER
- FULLOUTER
- INNER (default)

For more information, see "Working with Multiple Tables in a Master-Detail Hierarchy" in Developing Fusion Web Applications with Oracle Application Development Framework.

62.2.5 Understanding View Criteria Guidelines

A view criteria identifies filter information for the rows of a view object collection. Required filters for view objects for Oracle BI Applications should be created using named view criteria. This includes:

- Security filters
  For more information about security filters, see Understanding Business Intelligence Filters.

- Functional filters for Transactional Business Intelligence or Oracle BI Applications.
  Only filters required by both Transactional Business Intelligence and Oracle BI Applications should be created for view objects that are shared by both products.

- Filters to distinguish different logical entities based on the same entity object. (For single entity object view objects).

For more information, see "Working with Named View Criteria" in Developing Fusion Web Applications with Oracle Application Development Framework.

62.3 Understanding Oracle Business Intelligence Design Patterns

This section discusses Oracle Business Intelligence design patterns including flattened view objects, fact-dimension relationships, self referencing entities, filters, and translations.

62.3.1 Understanding Flattened View Objects

The grain of a fact table represents the most atomic level by which the facts may be defined. The fact or dimension grain required for Oracle Business Intelligence modeling should determine the flattening required for view objects. You should only create flattened view objects for fact or dimension levels required for either Transactional Business Intelligence or Oracle BI Applications. For example, if neither Transactional Business Intelligence nor Oracle BI Applications requires (purchase order) PO Shipments, then do not create a flattened POShipmentsVO.
When flattening entity objects in a view object, include only entity objects that do not change the grain of the fact or dimension. For example:

If attributes from a backing requisition line are needed on the POLinesVO, then the Requisition Line entity object should only be included in the flattened POLinesVO if the join does not change the grain of the POLinesVO to Requisition Line.

A 1:n relationship requires two view objects only if you want to aggregate attributes from the child and store the result at the grain of the parent.

Flattened view objects should be modeled in the Oracle Business Intelligence layer as a single logical table with multiple logical table sources.

62.3.2 Understanding Fact-Dimension Relationships

You should follow these rules when designing and creating fact and dimension view objects:

- Create separate view objects for fact entities and dimension entities.
- Do not flatten relationships between facts and dimensions into a single view object.
- Create a view link between the FactVO and the DimensionVO.
- Specify the FactVO as the source of the view link.
- Specify the DimensionVO as the target of the view link.

62.3.3 Understanding Self Referencing Entities (Self-Joins)

In the case of a fact view object where the self-joins represent two different but functionally important objects, you should create separate view object instances that represent the two objects. You should then define a view link between them.

If the self-join does not need to be represented as separate objects, you should resolve the Foreign Key ID column to a more meaningful column. For example:

The InvoiceheaderVO contains the following attributes:

- InvoiceId
- InvoiceNum
- TaxRelatedInvoiceId
- CreditedInvoiceId

If you decide that these should be modeled as three separate facts, then you create two additional view instances, TaxRelatedInvoicesVO and CreditedInvoicesVO, with view links to the InvoiceHeaderVO.

If you decide that they do not need to be modeled as separate objects, then you should create the two additional joins inside the InvoiceHeaderVO to bring in TaxRelatedInvoiceNum and CreditedInvoiceNum.

Row and Column flattening is required for view objects with self-joins that are modeled as dimensions in Oracle Business Intelligence Applications. You should determine the level of flattening required on a case-by-case basis.
62.3.4 Understanding Business Intelligence Filters

Only filters that are common to both Transactional Business Intelligence and Oracle BI Applications should be defined on shared view objects. If Transactional Business Intelligence requires additional filtering for a Transactional Business Intelligence specific application then it should be defined on the Oracle BI EE layer. If Oracle BI Applications needs to filter data from a shared view object for extraction, these filters need to be defined in the ETL layer.

Also note that view criteria cannot be pruned from the SQL at runtime.

62.3.5 Understanding Translations

All Fusion translatable entities with a corresponding _TL table require entity objects based on both _B and _TL entity objects. You should create a flattened view object to join _B and _TL entity objects.

Oracle BI Applications performs ETL (extract, transform, and load) processes from the flattened view object with no additional filters. However, Transactional Business Intelligence requires an additional session language filter in Oracle Business Intelligence layer.

Note:
The entity object associations required for ID and Code resolutions to Multi-Language Support-enabled entities should use a _VL view.

62.3.6 Understanding Date Effectivity

All date effective entities for a logical fact or dimension should be flattened and adhere to the following:

- Date effective entity objects and view objects should be marked as such according to Oracle ADF.
- Flattening requirement excludes scenarios where other design considerations require not flattening the entity objects in the view object. For example, 1:n relationships.
- Both entity objects are date effective.
- The PersonsVO should be flattened to include both PersonEO and PersonDetailEO and should also be marked as Date Effective.

In other words, there should be a single current person details record for each person record.

For more information, see "How to Store Data Pertaining to a Specific Point in Time" in Developing Fusion Web Applications with Oracle Application Development Framework.

62.3.6.1 About Date Effectivity Exceptions for Oracle BI Applications

Oracle BI Applications identifies any date effective entity objects from which historical information is needed; single view object flattening does not meet their requirements. To compensate, you need to:
• Create a separate view object for these entity objects. The entity object is removed from the flattened view object.

• Create view links to join these view objects.

• You must only include one historical entity object for any given view object in Oracle BI Applications.

You should still mark view objects as date effective so that Transactional Business Intelligence can share and date effective predicate can be applied in the Oracle Business Intelligence layer.

62.4 About Designing and Securing Fact View Objects

Separate view objects should be created for fact entities and dimension entities. Relationships between facts and dimensions should not be flattened into a single view object. Instead, you should create a separate FactVO and DimensionVO and then create a view link between them. Specify the FactVO as the source of the view link, and the DimensionVO as the target of the view link.

62.4.1 About Designing Fact View Objects

A flattened view object should be created for each logical fact grain in Transactional Business Intelligence and Oracle BI Applications. For example:

A purchase order contains four fact levels: Header, Lines, Shipments, Distributions. Flattened view objects should be created to represent each of the four fact grains, as shown in Table 62-2.

<table>
<thead>
<tr>
<th>Flattened View Objects</th>
<th>Fact Grains</th>
</tr>
</thead>
<tbody>
<tr>
<td>POHeaderVO</td>
<td>Header</td>
</tr>
<tr>
<td>POLinesVO</td>
<td>Header +Lines</td>
</tr>
<tr>
<td>POShipmentsVO</td>
<td>Header +Lines +Shipments</td>
</tr>
<tr>
<td>PODistributionsVO</td>
<td>header +Lines +Shipments +Distributions</td>
</tr>
</tbody>
</table>

Entity objects can be included in flattened view objects as required if the view object grain does not change.

---

**Note:**

View links are not required between these view objects.

---

**Join Type for Multi-Level Facts**

Join types on entity associations between multi-level facts should be inner joins. This is because there are some security impacts if entity associations are modeled as outer joins. For example:

To support the query “Show me all PO headers with no associated distributed rows”. If an outer join is used, you would need to implement security on both the header and the distribution entities in the DistributionVO. This would prevent pruning of the
header entity from the DistributionVO; it is also a change from current guidelines to only secure the primary entity.

### 62.4.2 About Securing Fact View Objects

The following are general guidelines for securing fact view objects. The sub-sections describe different design patterns that may arise for Oracle Business Intelligence use cases. Also included are solutions for each design pattern.

Fusion Data Security view criteria should be applied to the fact view object.

For more information about Fusion Data Security view criteria, see How to Secure Rows for Read_ Update and Delete Access.

The data security view criteria should contain:

- **Privilege** – Relevant object privilege.
- **Object** – Object being secured.
  
  For Multi-Organization Access Control (MOAC) style grants, the object being secured is Business Unit, based on the way MOAC grants are authored. For other grants, it can be the transactional object.
- **Alias** – Alias for object.
  
  Alias is mandatory for view objects for Oracle Business Intelligence Applications privileges.

For example, For Payment Invoices fact view object using Business Unit security (MOAC style), the privilege is:

"FNDDS__AP_MANAGE_PAYABLES_INVOICE_DATA__FUN_ALL_BUSINESS_UNITS_V__BU"

The fact view object requires an entity object for securing the table. (BU in this example). The join between the fact and the securing table should be properly resolved. The alias used in the view criteria should be that of the entity object corresponding to the **Object** in privilege (BU in this example).

If a non-MOAC grant is made for a transaction object, such as, for example, the Payment fact of the Fusion Incentive Compensation Management (ICM) Application, the object and alias refer to the ICM Payment entity. For example:

"FNDDS__VIEW_INCENTIVE_COMPENSATION_PAYSHEET_DATA__IC_INCENTIVE_COMPENSATION_PAYSHEET__ICPAY"

### 62.4.2.1 Secure the Same Transaction by Multiple Entities for Different Roles

In Oracle Fusion Applications, transaction data can be secured by more than one entity, based on the role used to access the transaction data. For example, consider the case of the Fusion Incentive Compensation Management (ICM) Application, in which:

- **Role Incentive Compensation Paysheet Management Duty** can see Incentive Compensation paysheets for the participants for whom they are responsible.
- **Role Incentive Compensation Process Management Duty** can see Incentive Compensation paysheets for the business units for which they are authorized.

In the above case, because the view object for the transaction object implements single data security privilege, the privilege should be able to provide access based on business unit as well as participants. Building this privilege provides a logical filter similar to:
You can achieve this by creating a new privilege and having two policies created using the same privilege. One policy should be created using instance set to provide "for business units for which they are authorized", and the second policy should be created using instance set to provide "for the participants for who they are responsible". The policies should be granted to existing roles.

**To secure transaction by more than one entity:**
The following steps are based on the Fusion ICM Paysheet use case.

1. Create a new data privilege titled View Incentive Compensation Paysheet Data.

2. Author the following data security policies, using existing duties and the new data privilege defined in Step 1:
   a. `<Incentive Compensation Paysheet Management Duty>` can `<view>` `<Incentive Compensation Paysheets>` `<for the participants for who they are responsible>`
   b. `<Incentive Compensation Process Management Duty>` can `<view>` `<Incentive Compensation Paysheets>` `<for business units for which they are authorized>`

3. Define the following grants:
   a. For the data security policy described in Step 2a:
      For this data security policy, you should attach the View Incentive Compensation Paysheet Data data privilege to the same FND_MENU that contains the grant for the Manage data privilege. This grants VIEW privileges to the same roles that have Manage privilege, reducing the number of grants to be managed.
   b. For the data security policy described in Step 2b:
      For this data security policy, you would create a non-MOAC grant against the Incentive Compensation Paysheet object against the business unit (BU) data role. This grant is parameterized instance set based, with the instance set returning Paysheet data by BU, using BU on the data role as the parameter value. This grant carries only the VIEW data privilege.

   **Note:**
   This is a data role grant and the role and grant is generated during the implementation phase using the data role template.

4. Use privilege View Incentive Compensation Paysheet Data in data security view criteria in Incentive Compensation Paysheet. The view criteria should be like Example 62-1 assuming IC_INCENTIVE_COMPENSATION_PAYSHEET is the object registered in FND_OBJECT and ICPAY is the alias used for the entity object.

**Example 62-1  Data Security View Criteria Example**

"FNDDS__VIEW_INCENTIVE_COMPENSATION_PAYSHEET_DATA__IC_INCENTIVE_COMPENSATION_PAYSHEET__ICPAY"
Caution:

With regards to the above proposal, if a user happens to have both Incentive Compensation Paysheet Management Duty and Incentive Compensation Process Management Duty roles granted; the Business Intelligence report will show the UNION of data, such as data for authorized business units AND data for responsible participants.

Whether such reporting behavior is acceptable should be decided on a case by case basis.

For Oracle BI Applications, the UNION effect for the above example, based on Oracle Fusion Incentive Compensation Management reporting (Access by Participants and access by business units), must be achieved in Oracle BI EE based on the OR join for individual dimensions. This can potentially be achieved by using two separate groups (one for business unit and another for participants) and having a user access to both groups (since predicates are ORed across Oracle BI EE groups).

There are other use cases that fall into same design pattern of a transaction being secured by multiple entities and Oracle Business Intelligence implementation needing UNION access. For example, in Oracle Fusion Projects, the transaction table Expenditure Item is secured by Business Unit as well as by Project. For Oracle Business Intelligence reporting, the query on Expenditure Item should return rows for authorized business units for a user as well as authorized project for user.

In general, while the Oracle Business Intelligence use cases for transaction being secured by multiple entities will be similar; application teams can make their own decisions about how they implement an Oracle Business Intelligence solution. For example, in the case of the Oracle Fusion Incentive Compensation Management and Oracle Fusion Projects applications, you can implement different solutions for achieving the same results by having different styles of grants and roles. Therefore, application teams should choose their own implementation based on their existing roles and privileges, and the approach they want to take for Oracle Business Intelligence solution.

62.4.2.2 Secure Transactions Different from Securing Dimensions

When transactions are analyzed in context of dimensions, sometimes the dimensions have their own security, which is not applicable for usage with the transaction.

For example, Grade data is secured using Fusion data security. When analyzing Assignment data, relevant information from the Grade dimension is required; however, the data security for the Grade dimension is not applicable when being used for analyzing assignments. Instead, the Grade dimension behaves as an unsecured source of data when used with assignment fact.

A solution for this use case is to create two view objects for the dimensions for which security is not required when analyzing fact. The two view objects should form two logical table sources (LTS) for the dimensions:

- The first dimension view object implements data security. This is used in dimension browsing and can include all columns required for dimension browsing. To ensure BI EE uses the secured version for dimension browsing, make sure it is higher up in the list of Logical Table Sources (LTS) than the unsecured one.
The second dimension view object should be unsecured. To ensure that the unsecured view object is used to combine with the fact, physical joins should be defined between the physical fact table and the physical table for the unsecured version of the dimension view object.

Caution:
Dimensions, for which unsecured view objects are created, may contain sensitive attributes. If this is the case, then you must make sure that the unsecured view object does not contain these sensitive attributes.

Dual (secured and unsecured) view objects are only required for entities that fall in this design pattern. Entities not requiring both secured and unsecured access do not require dual view objects.

62.4.2.3 Join Facts to Facts
Analysis of a fact may need reference information from another fact. In Transactional Business Intelligence, this is handled by creating a degenerate dimension for a fact whose attribute information is used in other facts. The degenerate dimension is just a logical layer entity in the RPD and it uses the same view object as the underlying fact. Consequently, the data security for degenerate fact is the same as that of underlying fact table.

This may create a problem when the degenerate dimension is used in another fact that has different security than the degenerate dimension (or more accurately, the fact underlying the degenerate dimension). For example:

- There is a degenerate dimension, Dimension A on top of Fact A.
- Dimension A is used in Fact B as a reference.
- Fact B is secured using a different dimension (or different privilege) than Fact A, which was used to source for Dimension A.

In such cases, the security of both Fact B and Fact A should be applied; whereas the desired result was just to apply security of Fact B.

62.4.2.4 Secure MOAC-Based transactional Applications
Multi-Org Access Control (MOAC) ADF infrastructure enables Fusion transaction applications to implement business unit based data security. Because the Oracle Business Intelligence technical stack works on the view definitions, the ADF Business Components MOAC infrastructure does not work for view objects for Oracle Business Intelligence Applications. These view objects should instead use underlying Fusion Data Security to support business unit based security.

62.5 About Designing and Securing Dimension View Objects
This section discusses how to design and secure dimensions.

62.5.1 Overview of Designing Dimension View Objects
A flattened view object should be created for each logical dimension grain in Transactional Business Intelligence and Oracle BI Applications. For example, for the Geography dimension, the following view objects are required to represent each dimension grain:
• Zip Grain — Zip Code
• City Grain — City + Zip Code
• State Grain — State + City + Zip Code

These should be modeled as a single Geography logical dimension table with multiple logical table sources, one for each of the dimension grains.

**62.5.2 About Designing Business Unit Dimensions**

Create a view link between the Transactional Business Intelligence fact view objects that have Business Unit dimensionality to the common business unit dimension view object based on Business Unit ID.

**62.5.3 About Securing Dimension View Objects**

If the dimension must be secured, then the FND view criteria should be applied on the dimension view object.

**62.5.3.1 About Securing Dimensions Composed of Multiple Entities**

The following use case, is an example of how you should secure dimensions that are composed of multiple entities.

The Dimension Inventory Organization is composed of the following three entities:

• InventoryOrgParameters
• HrOrganizationUnits
• HrLocations

Human Resources (HR) entities may have their own security. However, for the InventoryOrgParameters entity, only the security defined by inventory product Manage Inventory Org Parameters is necessary. In other words, data security on HR entities should be ignored when consumed in InvOrg.

This use case is similar to Securing Transactions Different from Securing Dimensions, where unsecured view objects are used for dimensions.

**62.5.3.2 About Securing Transactions Using Dimension with Dimension Browsing Unsecured**

The Dimension Business Unit is used to secure transaction data. When used in conjunction with transaction data, a secured version of the Business Unit, which can return business units allowed for a user for a function, is required. For example, a secured version of Business Unit is required to populate init block security variables for Oracle BI Applications.

However, if a user needs to browse only the business unit data, the user is allowed to see all dimensions. Therefore, it is deemed an unsecured dimension when dimension browsing in Oracle BI Applications. To use an unsecured view object for dimension browsing, make sure it is higher up in the list of LTSs than the unsecured one.

**62.5.4 About Using Multi-Valued Dimension Attributes**

Separate view objects should be created for primary dimension entity and multi-valued dimension attribute entities. For example:

Using the Person model, the following view objects should be created:
• PersonVO — Person Only
• PersonAddressesVO — Addresses Only
• PersonPhonesVO — Phones Only.

The following view links establish relationships between view objects:
• PersonToAddressesVL — PersonVO -> PersonAddressesVO
• PersonToPhonesVL — PersonVO -> PersonPhonesVO

Note:
The above example uses the Person model with a person having address and phone. Keep in mind that Transactional Business Intelligence models only the primary address and phone number while Oracle BI Applications can model more than one address and phone number per person.

62.5.5 About Using Junk Dimensions and Mini Dimensions

Junk dimensions should not be directly sourced from view objects. Oracle BI Applications should build them from Fact Stage tables. Transactional Business Intelligence should build them from the degenerate attributes in Fact tables.

Mini dimensions should not be sourced from view objects. Oracle BI Applications should build them from Dimension tables.

62.5.6 About Using Secured and Unsecured Dimension View Objects

There are several situations in which a secured dimension view object must be deployed with an accompanying unsecured dimension view object. In this case, the term unsecured does not simply mean that security is disabled, but also that a subset of the column set of the secured dimension view object may also be excluded from the unsecured version.

Generally, the strategy for developing and deploying a pair of corresponding dimension view objects, where one is secured and the other unsecured, consists of the following:

• A base dimension view object satisfying the basic, functional requirements for data retrieval is initially developed.

• The base dimension view object is used to create a secured dimension view object by using the methods and strategies described earlier in this section.

• An unsecured dimension view object is developed by manually creating an exact copy of the original base dimension view object.

  The unsecured dimension view object is named <VO Name> ListPVO, where <VO Name> is the name of the base dimension view object.

• The unsecured dimension view object is modified so as to exclude sensitive columns from its column sets.

  The unsecured dimension view object is deployed in the same application module as its associated secured dimension view object.
Consuming applications must build View Links to both the secured and unsecured dimension view object definitions. Once the secured and unsecured dimension view objects have been deployed, you can begin developing models based upon them in Oracle Business Intelligence.

62.6 About Designing Date Dimensions

This section discusses the gregorian calendar as well as the special handling that is required for fiscal calendar, projects calendar, Timestamp columns, and role-playing data dimensions.

62.6.1 About Using the Gregorian Calendar

Date dimension view objects for the gregorian calendar are delivered through the ATG libraries.

You should create a view link between the gregorian calendar day level view object and all the facts that join with the date dimension. Create the view link with the Fact view object as the source and the day level view object as the target.

For all other calendars needed for the fact in a particular functional area, a view link should be created to the time dimension at the day level of the fact. For example, if the fact is at day level in Financials and the reporting calendar is fiscal (in addition to gregorian), view links should be created to the day level of the fiscal calendar.

62.6.2 About Using the Fiscal Calendar

If the fact is at the day level, you should create view links to the day level of the fiscal calendar only.

For all facts at the day level, the view link between the Fact view object to the Day Level flattened view object should include the ADJUSTMENT_PERIOD_FLAG =N condition to avoid double counting if the same day belongs to a normal period as well as an adjusting period.

62.6.3 About Using the Projects Calendar

Projects facts that need to be analyzed by the fiscal calendar requires a view link between the fact and the day level of the fiscal calendar on the date. Also required is a view link between the fact and the General Ledger on the Ledger ID column using the Fun_all_business_units_V table that is present in the fact side.

Projects facts that need to be analyzed by the projects calendar requires a view link between the fact and the day level of the projects calendar on the date. Also required is a view link between the fact and the pjf_bu_impl_all_v table on the Business Unit Id.

62.6.4 About Using Timestamp Columns

If the date column of a fact view object involves timestamp then teams will need to create a new SQL derived attribute to populate the date without the timestamp. A view link will also need to be created using the new date column of the fact view object and day level time dimension view object.

If the fact view object date column does not have the timestamp then it can be used for creating the view link.
62.6.5 About Using Role-Playing Date Dimensions

If role-playing date dimensions are required, Transactional Business Intelligence is required to create aliases of the date view object in the Oracle BI EE physical layer. Duplicate view object instances should not be included in the model.

62.7 About Designing Lookups as Dimensions

If a lookup type is used as a dimension in Transactional Business Intelligence, you must deliver the dimension view object as follows:

- Create a `<Product short name>LookupsVO`.
- This new view object should be based on the product specific lookup view on `FND_LOOKUPS`.
  
  If a view to `FND_LOOKUPS` is not available or not required for online transaction processing, the view object should be based on `FND_LOOKUPS` directly with an additional filter on all included Application IDs for the product.

- View criteria should be created for each Lookup Type.
- Create a view instance using the above view criteria for each dimension.

Foreign keys to low cardinality lookups, such as `FND_LOOKUPS`, should not be resolved in fact or dimension view objects. These should be resolved in the logical layer through the lookup function.

Business Transactional Intelligence-only low cardinality lookups should be resolved using entity object associations based on a `_VL` view.

62.7.1 About Securing Data on Lookups

Lookup data can be striped by set ID. However, no use cases have been brought forward to date where the lookup data has been secured by explicit data security policies.

62.8 About Designing and Securing Tree Data

Application trees managed by Fusion tree management infrastructure may be exposed to Oracle Business Intelligence systems, such as Oracle BI EE, for analysis. This is done by providing a view object that contains a column-flattened version of tree data joined with tree data sources. Such a view object is called a `column-flattened view object for Business Intelligence (BICVO)`.

Designing and securing tree data for Oracle Business Intelligence involves the following activities:

- Designing a column-flattened view object
- Customizing the FND table structure and indexes
- Using declarative SQL mode to design view objects
- Securing Oracle ADF view objects for trees
62.8.1 About Designing a Column-Flattened View Object for Oracle Business Intelligence

Column-flattening is generally available for level-based trees. For those trees that may be exposed to Oracle Business Intelligence systems, such as Oracle BI EE, column-flattening for value-based trees also is available.

Figure 62-2 illustrates a generic example of a value-based tree.

Figure 62-2  Value-Based Tree Example

![Value-Based Tree Example](image)

Each node has a unique identity, in this case denoted by dot-separated numbers that correspond to the node’s relative ordering in the overall parent-child structure. Such value hierarchies may be arbitrarily recursive (in terms of recurring node types), and are usually ragged, or unbalanced. There is only a general concept of “level” in these hierarchies, which refers to the path distance (or depth) from the root node to some specified node.

Two nodes the same distance from the root are thought of as being at the same level. However, unlike true level-based trees, there is no requirement for nodes at the same level to possess a common set of properties. In fact, a node in a value-based tree may have any arbitrary collection of properties. When these trees are used to represent dimensional hierarchies, facts, metrics, or transactions, values may be joined to any node. There is no constraint that facts or transactions only be joined to lowest-level nodes, as is usually the case with level-based trees.

The example value-based tree shown in Figure 62-2, also has multiple top-level or root-level nodes. Since it has five levels (or equivalently, a maximum depth of four), a column-flattened representation of this tree requires a minimum of five columns. This is illustrated in Table 62-3.

Note:
In practice, you would never have single node trees. However, root nodes 2.0 and 3.0 are in Figure 62-2 to simply illustrate multiple top-nodes.
The following conventions apply to the logical column-flattened representation shown in Table 62-3.

- The first column (C0) contains a complete enumeration of each node in the tree. In this example, each node is represented by the value of its unique identity.

  Having the unique identity of each node of the hierarchy represented exactly once in the C0 column means that it is always possible to directly address each node, such as for purposes of joining with a transaction or measure, or for performing a calculation on that node.

- The last column (C4 in the example) always represents the root node of some rooted ancestral path of the tree.

- The intermediate ancestral path nodes between a given node in the C0 column and its ancestral root node in the C4 column, is represented by columns C1 through C3. Each column stores a reference to some node of the ancestral path, descending from C3 toward C0, filling each column (from right to left) with a reference to the next child node of the path. When a reference to the C0-th column node occurs, this reference is then repeated, if necessary, so as to pad the remaining columns until the C0 column is reached.

  Having the complete ancestral path, with unused columns padded toward the C0 node value, facilitates more efficient drill down operations.

- There is no implied ordering of the rows in the column-flattened representation. The complete hierarchy is represented by the table content, and a normalized representation can always be inferred or reconstructed from the flattened data set.

As far as Fusion tree management is concerned, the column-flattened representation always consists of several columns greater than, or equal to, the depth of the tree. If this were not the case, you would need a strategy for pruning or condensing the tree (for example, removal of intermediate nodes from the ancestral paths). However,
having the number of columns exceed the depth of the tree is never problematic, because of the repeated padding of C0 node values.

ATG services allows you to specify some fixed maximum depth of up to 32 levels when defining a tree. For example, if you specify a 20-level tree, your column-flattened representation will contain 20 columns, C0 through C19, with padding of values toward the leaf, as shown in Table 62-4.

Table 62-4  Column-Flattened Value-Based Tree Fixed at 20 Levels

<table>
<thead>
<tr>
<th></th>
<th>C15</th>
<th>C16</th>
<th>C17</th>
<th>C18</th>
<th>C19</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>1.2.1</td>
<td>1.2.1</td>
<td>1.2.1</td>
<td>1.2.1</td>
<td>1.2.1</td>
<td>1.0</td>
<td>2</td>
</tr>
<tr>
<td>1.2.1.1</td>
<td>1.2.1.1</td>
<td>1.2.1.1</td>
<td>1.2.1.1</td>
<td>1.2.1.1</td>
<td>1.0</td>
<td>3</td>
</tr>
<tr>
<td>1.2.1.1.1</td>
<td>1.2.1.1.1</td>
<td>1.2.1.1.1</td>
<td>1.2.1.1.1</td>
<td>1.2.1.1.1</td>
<td>1.0</td>
<td>4</td>
</tr>
<tr>
<td>1.2.2</td>
<td>1.2.2</td>
<td>1.2.2</td>
<td>1.2.2</td>
<td>1.2.2</td>
<td>1.0</td>
<td>2</td>
</tr>
<tr>
<td>1.2.2.1</td>
<td>1.2.2.1</td>
<td>1.2.2.1</td>
<td>1.2.2.1</td>
<td>1.2.2.1</td>
<td>1.0</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2.2</td>
<td>1.2.2.2</td>
<td>1.2.2.2</td>
<td>1.2.2.2</td>
<td>1.2.2.2</td>
<td>1.0</td>
<td>3</td>
</tr>
<tr>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>0</td>
</tr>
<tr>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>0</td>
</tr>
</tbody>
</table>

Think of the tree in Figure 62-2 as a true level-based tree, with fixed levels, single top-nodes, and all leaf nodes residing at the same lowest level of the tree (such as level zero, represented by column C0). In this case, you would actually have three separate trees, and the tree rooted at node 1.0 would have the logical column-flattened representation shown in Table 62-5, assuming the same “pad toward leaf values” scheme as with the value-based tree.

Table 62-5  Column-Flattened Level-Based Tree Rooted at Node 1.0

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>1.2.1.1</td>
<td>1.2.1.1</td>
<td>1.2.1.1</td>
<td>1.2.1.1</td>
<td>1.2.1.1</td>
<td>1.0</td>
</tr>
<tr>
<td>1.2.2.1</td>
<td>1.2.2.1</td>
<td>1.2.2.1</td>
<td>1.2.2.1</td>
<td>1.2.2.1</td>
<td>1.0</td>
</tr>
<tr>
<td>1.2.2.2</td>
<td>1.2.2.2</td>
<td>1.2.2.2</td>
<td>1.2.2.2</td>
<td>1.2.2.2</td>
<td>1.0</td>
</tr>
</tbody>
</table>

The notion of distance from the root is still relevant, even though all of the leaf nodes are assumed to reside at the same level (level zero, or C0).

62.8.1.1 How to Generate a BICVO Automatically Using Tree Management

Attributes from the column-flattened version of the tree data use standard ADF Business Components attribute naming conventions. Attributes from the tree data
sources also use the same naming convention, but are prefixed with DepN, where N is
the zero-based height of the node within the tree; for example, Dep7EmployeeName
or Dep13ProjectName. The Dep0 prefix is used for leaf nodes.

The following procedure is a summary of the overall process of defining and
generating declarative BICVOs for trees. For more detailed information about the
strategy for creating these BICVOs, see Guidelines for ATG-Registration and BICVO
Generation and About Securing ADF Business Components View Objects for Trees.

To generate BICVO automatically, do the following:

1. Ensure that the namespace path /oracle/apps/fnd/appcore/trees/
analytics is configured in Oracle Metadata Service (MDS). Example 62-2 shows a
sample MDS configuration.

2. Ensure that each view object attribute of the tree data source view objects is marked
as relevant to Oracle Business Intelligence (or not) via the BI Relevant property
that is exposed in the Property Inspector for the view object attribute.

   **Note:**
   By default, only primary key attributes are "BI Relevant". For performance
   reasons, it is recommended that only those attributes that are really relevant to
   Oracle Business Intelligence be marked as such to avoid generating very large
   BICVOs.

3. Ensure that column flattening is enabled by specifying the column-flattened table
and, optionally, the entity object for the table while setting up the tree structure.
For more information, see How to Create a Tree Structure.

   The tree management infrastructure then generates the BICVO for the tree
structure into MDS.

4. Secure the generated BICVO using the data security infrastructure. For more
information, see About Securing ADF Business Components View Objects for
Trees.

   The generated BICVO includes a special view criteria named FNDDS__BICVO. To
secure access to data through the BICVO, this view criteria must be enabled for
instances of the BICVO in any application module. At runtime, data security rules
affecting access to the tree data source view objects are automatically carried over
to the BICVO.

   **Note:**
   In Oracle Fusion Applications V1, only filter-based data security rules are
supported. In addition, only the "is descendant of" operator is supported.

**Example 62-2   MDS Configuration**

```xml
<adf-mds-config xmlns="http://xmlns.oracle.com/adf/mds/config"
    version="11.1.1.000">
  <mds-config version="11.1.1.000" xmlns="http://xmlns.oracle.com/mds/config">
    <persistence-config>
      <metadata-namespaces>
        <namespace path="/sessiondef" metadata-store-usage="mdsRepos"/>
        <namespace path="/persdef" metadata-store-usage="mdsRepos"/>
      </metadata-namespaces>
    </persistence-config>
  </mds-config>
</adf-mds-config>
```
62.8.2 About Customizing the FND Table Structure and Indexes

When using Oracle Fusion tree management to create and manage your trees, you should create and register its own, custom versions of the FND_TREE_NODE and FND_TREE_NODE_CF tables. This prevents applications from competing for use of the FND tables. Your custom tables must comply to the following rules:

- They must have custom, (preferably application-specific) names; for example, PJF_PROJ_ELEMENTS_CF is currently being used by the Projects team to implement a column-flattened table for the Task Hierarchy.

- The column names and column data types of each custom table must be the same as those of the corresponding FND table.

- Custom versions of FND_TREE_NODE_CF can define an index on each of the level-based foreign key references to support efficient drill-downs. However, it is understood that certain application query patterns do not necessitate this degree of indexing. Indexing is also not necessary if the column-flattened table is guaranteed to be relatively small.

- Custom versions of the FND_TREE_NODE_CF should not include the ENTERPRISE_ID column as part of the primary key index defined on the custom table. This is because this column is not currently used by Oracle Fusion tree management.

62.8.3 About Using Declarative SQL Mode to Design View Objects for Oracle Business Intelligence Applications

All view objects for Oracle Business Intelligence Applications should be constructed in declarative SQL mode. This ensures that correct SQL pruning can be applied to any composite view object incorporating the Oracle Business Intelligence view object. This requirement also applies to the BICVO generated by Oracle Fusion tree management. However, of all the possible configurations of ADF Business Components objects defining a tree data source, only two configurations in particular actually lend themselves to the generation of declarative-mode BICVOs by Oracle Fusion tree management.

These configurations have been formalized as two distinct design patterns:

- Design Pattern #1: Single data source view object, single data source entity object
• Design Pattern #2: Multiple data source view objects, unique data source view object per depth of tree, single data source entity object per data source view object

Although either pattern can be used in the realization of either tree type, the first pattern is generally better suited to value-based trees, while the second pattern is more natural for level-based trees. However, the patterns are aimed primarily at supporting the automated generation of declarative-mode BICVOs, rather than supporting either particular type of tree.

62.8.3.1 About Using Single Data Source View Object Design Pattern

This pattern ensures that Oracle Fusion tree management can generate a declarative-mode BICVO from an Oracle Applications Technology (ATG)-registered data source. Figure 62-3 illustrates the ADF Business Components object configuration defining declarative BICVO pattern #1.

Figure 62-3 Declarative BICVO Based on Single Data Source View Object, Single Data Source Entity Object

In this pattern, there is a single data source entity object and a single data source view object based on that entity object. The data source view object is a declarative-mode view object built by developers and registered with Oracle Fusion tree management. The data source entity object in turn is based on a _VL database view that joins the data source base table (_B) with a table of translated values (_TL).

A second entity object is defined for the column-flattened table. Currently, the column-flattened table entity object must be created manually and made known to the generated BICVO via a manual workaround. Additionally, a collection of entity object associations, each joining the column-flattened entity object with the data source entity object for a unique level or depth of the tree, must also be created manually. If the application design requires that the base data source table expose multiple entity objects for any reason, then a _VL database view must be defined to join the multiple entity objects (possibly along with any translated attribute values), and that _VL database view must support the single data source entity object.

Once the data source view object is registered with Oracle Fusion tree management as part of the tree structure definition process, and the required manually-created objects
are all in place, a declarative BICVO may then be generated by Oracle Fusion tree management.

This declarative-mode BICVO pattern is well-suited for value-based trees, since value-based trees are most often represented at the data source level by a single table with a recursive self-join. However, there is nothing about the pattern that strictly requires its use in value-based hierarchies, nor prohibits its use from other types of hierarchies (such as level-based or hybrid). The primary objective of this pattern is to facilitate the automatic generation of a declarative-mode BICVO from an ATG-registered tree.

62.8.3.2 About Using Multiple Data Source View Objects Design Pattern

This pattern ensures that Oracle Fusion tree management can generate a declarative-mode BICVO from an ATG-registered tree. Figure 62-4 illustrates the ADF Business Components object configuration defining declarative BICVO Pattern #2.

Figure 62-4  Declarative BICVO Based on Multiple Data Source View Objects, Unique Data Source View Object per Level, Single Data Source Entity Object per Data Source View Object

In this pattern, there are multiple data source view objects, with a unique data source view object representing each level or depth of the tree. Each data source view object is based on a single, unique data source entity object. Each data source view object is a declarative-mode view object built by developers and registered with Oracle Fusion tree management. All of the data source view objects must be declarative-mode view objects; otherwise, a declarative-mode BICVO can not be generated. As with the previous pattern, each data source entity object in turn is based on a _VL database view that joins some data source base table (_B) with a table of translated values (_TL). While multiple _VL database views are represented in the diagram, there is no hard-and-fast requirement that each data source entity object actually be built on top of a unique _VL database view. The diagram simply admits the possibility of multiple such views, presumably one per level or depth of the tree.

The same as with design pattern #1, an entity object is also defined for the column-flattened table, and must also be created manually, and is made known to the generated BICVO via a manual workaround. This column-flattened table entity object is also joined to the data source entity objects via a collection of entity object associations. However, each entity object association relates the column-flattened table entity object to a unique data source entity object representing a particular level or depth of the tree.

If the application design requires that the base data source table expose multiple entity objects per tree level or depth, then a _VL database view must be defined to join the
multiple entity objects (possibly along with any translated attribute values) at that tree level or depth, and that _VL database view must support the single data source entity object for that tree level or depth.

Once the data source view objects have been registered with Oracle Fusion tree management as part of the tree structure definition process, and the required manually-created objects have all been put in place, a declarative BICVO may be generated by Oracle Fusion tree management.

This declarative-mode BICVO pattern is well-suited for level-based trees, since level-based trees are often built on top of multiple data sources, with a unique data source per level. However, there is nothing about the pattern that strictly requires its use in level-based hierarchies, nor prohibits its use from other types of hierarchies (such as value-based or hybrid). The primary objective of this pattern is to facilitate the automatic generation of a declarative-mode BICVO from an ATG-registered tree.

62.8.3.3 About Setting the Declarative-Mode BICVO Properties

To ensure correct SQL pruning, you must set the property values of the generated declarative-mode BICVO as follows:

- Designate the column-flattened table entity object as the primary entity of the BICVO.
- Designate the data-source entity objects as secondary or reference entities of the BICVO.
- Do not mark primary key attributes of the data source entity objects as primary-key attributes in the resulting column-set. (These are exposed by the generated BICVO).
- Set the `selectedInQuery` property of any non-primary key attribute of the generated BICVO to `false`.

62.8.4 Guidelines for ATG-Registration and BICVO Generation

The Oracle Fusion Applicationsteam that owns the tree is responsible for creating a custom tree node (parent-child relationship) table that is structurally equivalent to `FND_TREE_NODE`. Once the tree node table has been created, it is registered with ATG via the Oracle Fusion tree management tree creation UI.

The Oracle Fusion Applicationsteam is also responsible for creating a custom column-flattened table that is structurally equivalent to `FND_TREE_NODE_CF`. This custom table is depicted in Figure 62-3. Once created, it is also registered with Oracle Fusion tree management as the column-flattened table associated with the tree in the Oracle Fusion tree management creation UI.

The Oracle Fusion Applicationsteam must then create both data source view objects and associated data source entity objects, according to either of the structural patterns illustrated in Figure 62-3 and Figure 62-4. As with the tree node and column-flattened tables, the data source view object is also registered with Oracle Fusion tree management, via the Oracle Fusion tree creation UI. During the registration process, the developer may specify a custom property on any of the data source columns, indicating to Oracle Fusion tree management that these columns are relevant to Oracle Business Intelligence and need to be exposed at each level within the BICVO. This collection of Oracle Business Intelligence attributes is represented by the set of view attributes attached to the data source view object. Consequently, the generated BICVO will join these columns in from the data source entity object at each level of the tree, immediately following the level-specific data source foreign key references; that is, the
sequence of DEP*_PK* columns are followed by a set of columns representing each of
the BI-relevant attributes.

In addition to view attributes representing the Oracle Business Intelligence-relevant
columns of the data source, the data source view object may also be configured with
one or more view criteria filters. In particular, a view criteria must be defined to
enforce data security if there’s a requirement for data security at the source level. Any
other relevant filters required by reporting may also be specified and attached to the
data source view object. Each of these view criteria must specify a logical AND
condition as its connective to other defined view criteria.

Next, using the Oracle Fusion trees creation UI, the developer automatically generates
the BICVO; that is, the column-flattened BICVO based on the column-flattened table.
In Figure 62-3 and Figure 62-4, dashed lines represent joins on the underlying entities
that are automatically added by Oracle Fusion tree management to the BICVO
definition at runtime. These joins are inferred by ATG internal generation logic via
inspection of the data source view object and its attendant view attributes and view
criteria, as well as inspection of the registered column-flattened table.

The BICVO, as generated by Oracle Fusion tree management, also includes a
placeholder view criteria that is otherwise empty and specifies a logical OR condition as
its connective to any other view criteria that might be defined as part of the BICVO.
This placeholder view criteria is defined for data security purposes, and at the current
time, simply directs ATG logic to invoke the data security view criteria defined on the
data source view object.

---

**Note:**

There may also be a requirement to supply Oracle Data Integrator (ODI) with
translations via a view object that is separate from the base data source table
or _VL database view. In this case, you must develop a view object and entity
object pair that directly goes against the translations table (_TL).

---

You must take this entire collection of ADF Business Components objects, both hand-
crafted and generated alike, and package them for deployment as part of an
appropriate application module. Note that any ATG-generated artifacts, such as the
BICVO, is generated to reside within the Oracle Fusion Middleware Extensions for
Applications package namespace, which is:

`oracle.apps.fnd.bi.applicore.trees.bi.model.view`

Most of the Oracle Business Intelligence view objects and other artifacts are packaged
under the Oracle Business Intelligence analytics namespace, which is:

`oracle.apps.<LBATop>.<LBACore>.publicview.analytics`

However, the Oracle Fusion Middleware Extensions for Applications package
namespace is acceptable for ATG-generated objects seeing as they are artifacts of the
ATG-Oracle Fusion Middleware Extensions for Applications services infrastructure. If
the interfaces of these objects are publicly visible, this should not present any
problems to clients of these objects.

### 62.8.5 Guidelines for Hierarchy Depth and Conformance

It is possible for an inconsistency to arise between the three realizations of a particular
application hierarchy across the application, and the Transactional Business
Intelligence and Oracle BI Applications technologies.
Hierarchies on the Oracle BI EE server are necessarily limited to a maximum of 15 levels. However, Oracle BI Applications uses data warehouse tables to represent these hierarchies, and although the tables are not inherently bounded in size, restrictions on the number of levels of a given hierarchy being imported into the data warehouse are enforced by the ETL process. The majority of Oracle BI Applications hierarchies are fixed at eight levels plus a top-level for a total of nine fixed levels. A very small number of Oracle BI Applications hierarchies have greater than eight levels, plus a top-level and a base-level, with the largest of these hierarchies consisting of 21 fixed levels.

Trees, especially value-based trees, are generally unbounded in size. However, trees that have been implemented using Oracle Fusion tree management services are limited to 32 levels by the ATG infrastructure.

Problems can potentially arise when an application tree exceeds 15 levels. When this occurs, the corresponding Oracle BI EE representation of the tree, such as a Oracle Business Intelligence hierarchy stored within the repository (RPD), must be compressed to 15 levels. This is accomplished by retaining the leaf-level of the source tree (base-level), as well as the root-level (top-level), and pruning the tree starting with the base-1 level and working up the tree until enough levels have been removed.

Table 62-6 illustrates the general mapping of levels on the Oracle Business Intelligence hierarchy to levels or depths of application trees. The logical representation of the application tree is expressed in terms of the columns of the column-flattened Oracle Business Intelligence view object for that tree, which has a maximum of 32 levels. In this case the 15-level (maximum) Oracle Business Intelligence RPD representation of the hierarchy is mapped to the 32-level (maximum) application BICVO representation of the tree by pruning the levels of the source tree designated by columns C1 through c17 of the BICVO.

<table>
<thead>
<tr>
<th>BI (RPD)</th>
<th>Application (BICVO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top</td>
<td>C31</td>
</tr>
<tr>
<td>Level Top +1</td>
<td>C30</td>
</tr>
<tr>
<td>.....</td>
<td>.....</td>
</tr>
<tr>
<td>Level Base - 1</td>
<td>C18</td>
</tr>
<tr>
<td>Base</td>
<td>C0</td>
</tr>
</tbody>
</table>

When mapping application trees to Oracle Business Intelligence hierarchies, there are two types of problems that may arise:

- The application tree exceeds 15 levels and the Transactional Business Intelligence realization of the hierarchy (provided by the Oracle BI EE server) has been pruned to 15 levels. However, the Oracle BI Applications realization of the hierarchy (provided by the ETL process) is allowed to exceed 15 levels. In this case, the Transactional Business Intelligence and Oracle BI Applications realizations of the hierarchy have different resolutions at their lowest levels.

- The application tree exceeds 15 levels and the Transactional Business Intelligence and Oracle BI Applications realizations of the hierarchy are both pruned to 15 levels. In this case, Transactional Business Intelligence and Oracle BI Applications
are the same in terms of resolution, but the Oracle Business Intelligenceside and the application side are not the same. For example, the application tree has greater resolution than its Oracle Business Intelligence counterpart.

The following are two possible consequences that may result from the problems outlined:

- Loss of information (loss of resolution) resulting from the pruning away of several lower levels of the hierarchy, as well as potential differences in information (resolution) between Transactional Business Intelligence and Oracle BI Applications.

- An effect on fact-based security at the pruned levels. For example, you have established certain privileges on facts joined to nodes at tree levels that are ultimately pruned away. The security privileges of facts that had been joined to the pruned nodes may have been more restrictive than those at ancestral levels.

62.8.5.1 Resolve Problems

There are basically two choices for either completely resolving, or at least mitigating, the potential problems.

**Note:**

Neither of the following resolutions require any actual implementation work. However, they do require a combination of policy and documentation.

- **Complete Resolution:**
  
  Any application tree that has a realization on the Oracle Business Intelligenceside (Transactional Business Intelligence or Oracle BI Applications) must be restricted to no more than 15 levels.

- **Mitigation:**
  
  Ensure that, if any application tree exceeds 15 levels, and that tree has realizations on both Transactional Business Intelligence and Oracle BI Applications, that both technologies maintain pruned realizations of this tree and have the same number of levels (such as 15 or less).

  For this resolution, it will be necessary that these situations be investigated and documented on a case-by-case basis. You must decide how you want to adjust the security privileges of metrics that had previously been joined to the pruned levels, and then revise your Oracle Business Intelligence models accordingly.

62.8.6 About Securing ADF Business Components View Objects for Trees

Data security privileges are effectively applied to the column-flattened representation of the tree (as described in Table 62-3) in the form of a filter based on an OR condition on the columns. For example, a reporting client has viewing privileges on nodes 1.1 and 1.2.2. This means that any row that contains either node in any of its columns (at any level in the tree) is viewable to the client, but the other rows are not. The viewable rows are shown in Table 62-7 in bold.
### Table 62-7  Column-Flattened Result Set with Data Security

<table>
<thead>
<tr>
<th>C0</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>1.2.1</td>
<td>1.2.1</td>
<td>1.2.1</td>
<td>1.2</td>
<td>1.0</td>
<td>2</td>
</tr>
<tr>
<td>1.2.1.1</td>
<td>1.2.1.1</td>
<td>1.2.1</td>
<td>1.2</td>
<td>1.0</td>
<td>3</td>
</tr>
<tr>
<td>1.2.1.1.1</td>
<td>1.2.1.1</td>
<td>1.2.1</td>
<td>1.2</td>
<td>1.0</td>
<td>4</td>
</tr>
<tr>
<td>1.2.2</td>
<td>1.2.2</td>
<td>1.2.2</td>
<td>1.2</td>
<td>1.0</td>
<td>2</td>
</tr>
<tr>
<td>1.2.2.1</td>
<td>1.2.2.1</td>
<td>1.2.2</td>
<td>1.2</td>
<td>1.0</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2.2</td>
<td>1.2.2.2</td>
<td>1.2.2</td>
<td>1.2</td>
<td>1.0</td>
<td>3</td>
</tr>
<tr>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>0</td>
</tr>
<tr>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>0</td>
</tr>
</tbody>
</table>

If the use of the `DescendantOf` hierarchical referencing operator is also available, enabling the display of rows that contain either 1.1, 1.2.2, or any descendant of either of these two nodes, then the viewable rows include the rows that are displayed in bold in Table 62-8.

### Table 62-8  Column-Flattened Result Set with Data Security and DescendantOf Operator

<table>
<thead>
<tr>
<th>C0</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.2</td>
<td>1.0</td>
<td>1</td>
</tr>
<tr>
<td>1.2.1</td>
<td>1.2.1</td>
<td>1.2.1</td>
<td>1.2</td>
<td>1.0</td>
<td>2</td>
</tr>
<tr>
<td>1.2.1.1</td>
<td>1.2.1.1</td>
<td>1.2.1</td>
<td>1.2</td>
<td>1.0</td>
<td>3</td>
</tr>
<tr>
<td>1.2.1.1.1</td>
<td>1.2.1.1</td>
<td>1.2.1</td>
<td>1.2</td>
<td>1.0</td>
<td>4</td>
</tr>
<tr>
<td>1.2.2</td>
<td>1.2.2</td>
<td>1.2.2</td>
<td>1.2</td>
<td>1.0</td>
<td>2</td>
</tr>
<tr>
<td>1.2.2.1</td>
<td>1.2.2.1</td>
<td>1.2.2</td>
<td>1.2</td>
<td>1.0</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2.2</td>
<td>1.2.2.2</td>
<td>1.2.2</td>
<td>1.2</td>
<td>1.0</td>
<td>3</td>
</tr>
<tr>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td>0</td>
</tr>
<tr>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>3.0</td>
<td>0</td>
</tr>
</tbody>
</table>
Note:

The generalized OR filter can be restricted. For example, to apply only to the C0 column. This ensures that only nodes and optionally their descendants, for which a client has sufficient privileges, are viewable from the column-flattened result set.

62.8.6.1 About Security Implementation

The base table view object and column-flattened view objects (BICVO) are separate view objects. However, the data security definition must be consistently applied to both the base table view object and BICVO. For example, BICVOs must not have different data security behavior than the base entity on which security has been defined by Oracle Fusion Applications. This is achieved by Fusion tree management using the following process:

• When Oracle Fusion tree management generates the BICVO, it automatically adds an Oracle Fusion data security view criteria to the BICVO (FNDDS__BICVO). You must not change this view criteria’s name but must ensure that it is enabled for the application module for the Transactional Business Intelligence.

• The view criteria predicate for BICVO is generated from the base table view object at runtime by Oracle Fusion tree management. This ensures that BICVO data security is in sync with the base object.

• The following restrictions are placed on the base object view criteria so that the base view criteria is mapped to the BICVO view criteria, (which may have different column names), at runtime:

  – The base object view criteria must only use "Filter", which stores predicates using metadata. It cannot use SQL.

  – The base object view criteria must only use the DescendantOf hierarchy operator. It must not use any other hierarchy operators.

There may be situations in which a tree must support both secured and unsecured access. In this case, the BICVO that exposes the tree structure is deployed as both secured and unsecured versions.

The generated BICVO already has a security mechanism associated with it that is based on its data source view object. An unsecured version of the BICVO can be created by manually making a copy of the generated BICVO and editing it to exclude sensitive columns. Then, secured access to this edited BICVO is turned-off by de-activating the dummy FNDDS__BICVO view criteria associated with the BICVO. This causes the data source security view criteria to not be enforced. Again, both the secured and unsecured versions of the BICVO for the tree are to be deployed together in the same application module.

62.9 About Supporting Flexfields for Oracle Business Intelligence

You must do the following to allow the Flexfields ADF Modeler to generate a flattened view object containing only those attributes marked as BI Enabled:

• Set the BIEnabledFlag for your key flexfield

• Set the BIEnabledFlag for your descriptive flexfield
• Create Flexfield business components
• Define custom properties on the Oracle Business Intelligence application module
For information on how to perform these tasks, see the following sections in this book:
• Preparing Descriptive Flexfield Business Components for
• How to Prepare Key Flexfield Business Components for

62.10 About Supporting SetID

To properly resolve meanings for set-enabled attributes, the setID attribute must be exposed to the Oracle Business Intelligence layer. The setID attribute should be exposed using the appropriate method for the following reference types:

• Set-enabled lookups
• Set-enabled reference tables

62.10.1 How to Expose the SetID Attribute for Set-Enabled Lookups

The setID is required to retrieve appropriate meaning if the lookup is set-enabled. The Set Assignments Query is required to retrieve the setID.

To expose the setID attribute, do the following:

Set-enabled lookups (shared and Transactional Business Intelligence) are registered as warehouse domains and the SetAssignment entity object is already provided by ATG.

1. Build an entity object association between the Fact entity object and the SetAssignment entity object for each set-enabled lookup on the fact.

2. Expose setID as an attribute on the FactVO for each set-enabled lookup type on the FactVO.

The Lookup function is used to retrieve the translated meaning from the warehouse using setID parameter.

62.10.2 How to Expose the SetID Attribute for Set-Enabled Reference Tables

The setID is stored on set-enabled reference tables. A Unique ID is used as the primary key of the reference table; ID and language form the unique key of the translated reference table. The determinant value is not stored on the reference table; the foreign key used to reference the table is stored on transaction tables.

To expose the setID attribute, do the following:

Because the foreign key to the reference table already exists on the transaction, meanings for set-enabled attributes should be resolved depending on usage.

• Transactional Business Intelligence only:
  Resolve meaning on the base view object using entity object association, bringing in the setID attribute.

• Warehouse domain:
  A separate view object is required. Build a view link from the base view object to the reference view object. The setID attribute exists on the reference table view object.
62.11 About Supporting Multi-Currency

Oracle Fusion Middleware Extensions for Applications provides special MLS Currency view objects for Oracle Business Intelligence.

To support multi-currency, create view links from the primary entity currency code fields on transaction view objects to the new currency view object.
Implement ADF Desktop Integration

This chapter describes how to combine third party desktop applications with Oracle Fusion web applications.

The chapter includes these sections:

- Oracle Application Development Framework Desktop Integration Standards and Guidelines
- Skinning Excel ADF Desktop Integration Workbooks
- Configuring the WebLogic Server Frontend

63.1 Oracle Application Development Framework Desktop Integration Standards and Guidelines

ADF Desktop Integration makes it possible to combine third party desktop productivity applications with Oracle Fusion web applications, so you can use a program like Microsoft Excel as an interface to access Oracle Fusion web application data. Currently, ADF Desktop Integration supports using an Excel workbook to access descriptive and key flexfield data in your application.

ADF Desktop Integration is intended to provide integrated access across a variety of Oracle Fusion products from a variety of third-party interfaces. Consequently, it is important that you apply consistent standards for deployment and for look and feel to your implementation of ADF Desktop Integration.

For more general information about integrating Oracle Fusion web applications with desktop applications, see the Oracle Fusion Middleware Developing Applications with Oracle ADF Desktop Integration guide.

For information about using ADF Desktop Integration technology with flexfields, see Accessing Descriptive Flexfields from an ADF Desktop Integration Excel Workbook and How to Access Key Flexfields from an ADF Desktop Integration Excel Workbook.

Caution:

The Desktop Integration Developer’s Guide does not make explicit reference to technologies documented in this book, and this book does not repeat the content in the Desktop Integration Developer’s Guide, so you must read the Desktop Integration Developer’s Guide for a full understanding of how to use ADF Desktop Integration technology in general.
Standards for Naming and Organization

ADF Desktop Integration projects have many of the same code artifacts as standard ADF application UI projects. As such, the directory and naming standards for ADF Desktop Integration projects already have a good example to follow.

ADF Desktop Integration artifacts shall follow the same set of directory and naming standards as core ADF UI artifacts. However, you are also encouraged to distinguish your business component names as appropriate. For example, for the General Ledger Journal Entry functionality, the core ADF page has an application module called "JournalEntryAM." For the ADF Desktop Integration Journal Entry, the application module should be given a name similar to "DesktopJournalEntryAM" to prevent duplicate names.

63.1.1 How to Structure the ADF Desktop Integration Directories

The way that you code backing beans, page definitions and JSPX pages for ADF Desktop Integration will be different from the way you code core ADF UIs. For example, JSPX dialogs for ADF Desktop Integration require JavaScript. It is desirable to be able to distinguish between core UI artifacts and UI artifacts used by ADF Desktop Integration workbooks. Therefore, instead of checking in ADF Desktop Integration web picker artifacts into a "ui" package, it is recommended that you instead use a "di" (Desktop Integration) folder. Note that "di" and "ui" are the same length, so path string lengths will not change.

Following is an example of such a directory structure:

```
|-- adfmsrc
   |-- META-INF
       |-- <various generated files>
       |-- oracle
           |-- apps
               |-- <LBA Top>
                   |-- <LBA Core>
                       |-- di

|-- public_html
   |-- oracle
       |-- apps
           |-- <LBA Top>
               |-- <LBA Core>
                   |-- di
```

A major benefit to putting ADF Desktop Integration web picker dialog artifacts into a di folder is that automated standards checks can easily distinguish Desktop Integration-related objects and adjust their logic as needed. The directory structure under di will be organized the same way as the directory structure under the ui folder. For example, the bean, controller, page and util folders under di will be found in the same relative locations as the equivalent folders under ui.

The Excel Microsoft Office Open XML Format (XLSX) workbooks (and XLSM files, if required) should be checked into an excel folder within the public_html directory structure:

```
|-- public_html
   |-- oracle
       |-- apps
           |-- <LBA Top>
```

Oracle Application Development Framework Desktop Integration Standards and Guidelines
Example 63-1 shows some full directory paths for the ADF Desktop Integration artifacts on a Windows system, for a project in a leaf LBA called desktopJournalEntry. The folders contain page definitions, Excel and JSPX files, and beans, respectively:

Example 63-2 shows some directory paths for the ADF Desktop Integration artifacts in source control, for a project in a leaf LBA called desktopJournalEntry. The folders contain page definitions, [excel] and JSPX files, and beans, respectively:

Example 63-1  Directory Structure in Windows

D:\FinDashboardPrototype\gl\components\journals\desktopJournalEntry\di\adfmsrc\oracle\apps\financials\generalLedger\journals\desktopJournalEntry\di\pageDefs

D:\FinDashboardPrototype\gl\components\journals\desktopJournalEntry\di\public_html\oracle\apps\financials\generalLedger\journals\desktopJournalEntry\di\[excel|page]

D:\FinDashboardPrototype\gl\components\journals\desktopJournalEntry\di\src\oracle\apps\financials\generalLedger\journals\desktopJournalEntry\di\bean

Example 63-2  Directory Structure in Source Control

scs/gl/components/journals/desktopJournalEntry/di/adfmsrc/oracle/apps/financials/generalLedger/journals/desktopJournalEntry/di\pageDefs


scs/gl/components/journals/desktopJournalEntry/di/public_html/oracle/apps/financials/generalLedger/journals/desktopJournalEntry/di\bean

scs/gl/components/journals/desktopJournalEntry/di/src/oracle/apps/financials/generalLedger/journals/desktopJournalEntry/di\bean

63.1.2 How to Name Your ADF Desktop Integration Files

Because users can download workbooks to their desktops, providing each workbook with a unique name across applications is advisable. Make an effort to incorporate either the name of your product or the name of the primary Logical Business Object (LBO) involved, to create a meaningful name for your workbook. For example, the expenses workbook could be called ExpensesEntry.xlsx; the General Ledger journal entry workbook could be called JournalEntry.xlsx. However, because there is no current runtime or release requirement for unique names, the ADF Desktop Integration team will not coordinate this.

There is an open question regarding which workbook to source control and release: the design time version or the published version. For now, assume that both versions are source controlled. To that end, the published version should have the final name and the design time version should include the suffix "DT" in its filename. So in the expenses example, two workbooks will be source controlled: ExpensesEntry.xlsx and ExpensesEntryDT.xlsx, where the former is the published version and the latter is the design time version.
63.1.3 How to Implement the Dialog Attributes Declarative Component

The main function of the Dialog Attributes declarative component is to render the ADFDi reserved elements (ADFdi_CloseWindow, ADFdi_AbortUploadOnFailure and ADFdi_DownLoadAfterUpload) in the ADF pages that are used as dialogs in the spreadsheet.

This component replaces the ADFdi_CloseWindow, ADFdi_AbortUploadOnFailure and ADFdi_DownLoadAfterUpload span tags/outputText/JavaScript elements used in JSPX pages to render them as DI dialogs.

Since the component needs to render the value of the span tags (such as ADFdi_CloseWindow) based on the user's use case, it takes as an input the values to be rendered for these tags. To implement this, three properties are exposed on the component, one for each tag (ADFdi_CloseWindow, ADFdi_AbortUploadOnFailure and ADFdi_DownLoadAfterUpload). These properties need to be set based on how the page is used as a dialog in the spreadsheet. An overview of the properties is described in Table 63-1.

Table 63-1  Tag Properties Exposed on the Declarative Component

<table>
<thead>
<tr>
<th>Component Property</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>closeWindowBinding</td>
<td>String</td>
<td>This property maps to the DI Dom element: ADFdi_CloseWindow. The value supplied for this property will be set as the value of the ADFdi_CloseWindow tag. You can bind the property to a backing bean. Example: closeWindowBinding=&quot;#{SubmissionOptionsBean.dialogResult}&quot; where dialogResult is a string that is defined in a backing bean and the value is set when the OK or Cancel button is clicked.</td>
</tr>
<tr>
<td>downloadAfterUpload</td>
<td>Boolean</td>
<td>This property maps to the DI Dom element: ADFdi_DownLoadAfterUpload. This value usually is a constant, such as true or false. It appears in the component property inspector as a list from which you can choose a true/false value. This property must be specified only if you are using the dialog as a custom upload dialog. Example: downloadAfterUpload=&quot;false&quot;</td>
</tr>
<tr>
<td>abortUploadOnFailure</td>
<td>Boolean</td>
<td>This property maps to: ADFdi_AbortUploadOnFailure. This value usually is a constant, such as true or false. It appears in the component property inspector as a list from which you can choose a true/false value. This property must be specified only if you are using the dialog as a custom upload dialog. Example: abortUploadOnFailure=&quot;false&quot;</td>
</tr>
</tbody>
</table>
63.1.3.1 Add the Component to Your Page

Before you can add the component to a web page, you need to add its containing JAR file as a library reference.

To add the JAR file as a library reference:

1. Create a file system connection.
   - Enter the connection name, such as DI Components, and the directory path as /ade/<view name>/fusionapps/jlib.
   - Test the connection to make sure it is valid. If it is, click OK.

2. Select your user interface project in the application navigator pane.

3. Expand the connection you just created (FileSystem > <connection name>).

4. Right-click the library AdfFinFunPublicDeclarativeComponentsDi.jar and select Add to Project.

5. To make sure the library was added, open Project Properties > Libraries and Classpath > ADF Library. You should see an entry for AdfFinFunPublicDeclarativeComponentsDi.jar.

When you add the library as a library reference to your project, the component palette will contain the Di Components option that lists all the DI components available for use.

To add the component to the web page:

1. In the component palette (View > Component Palette), select the DialogAttributes component and drag it onto the desired web page.

   The DI component namespace is added to the jsp:root tag at the top of your page:


   The component is rendered as:

   `<di:dialogAttributes id="da1" />

2. Set the component attributes.
You can view the available properties on this component by looking at the property inspector. Set the properties as required. The component tag will look like:

```xml
<di:dialogAttributes id="da1"
abortUploadOnFailure="false"
downloadAfterUpload="false"
closeWindowBinding="#{SubmissionOptionsBean.dialogResult}"/>
```

You are now ready to run your project.

### 63.2 Skinning Excel ADF Desktop Integration Workbooks

Oracle Fusion ADF Desktop Integration workbooks share a common set of style definitions. This enables them to easily apply required changes to the look and feel. The various style definitions that are needed by applications developers are defined in a common Excel styles template file, which is the accurate source for ADF Desktop Integration styles in Oracle Fusion. Figure 63-1 shows an example of the ADF Desktop Integration styles in use; keep in mind that the look and feel is subject to change at any time.

*Figure 63-1 Example of ADF Desktop Integration Look and Feel in Excel*

### 63.3 Configure the WebLogic Server Frontend

When ADF Desktop Integration sends a spreadsheet to the client, it embeds the server public callback address. Then, when the spreadsheet is opened, it is able to authenticate and perform operations on the address.

To set the Frontend URL for the Administration Console:

1. Log in to Oracle WebLogic Server Administration Console.
2. Click *Lock & Edit*.
3. Expand the Environment node in the Domain Structure window.
4. Click *Servers*. The Summary of Servers page displays.
5. Select *Admin Server* in the Names column of the table. The settings page for AdminServer(admin) is displayed.
6. Click the Protocols tab.
7. Click the HTTP tab.
8. Set the Front End Host field to admin.mycompany.com (your LBR address).

9. Save and activate the changes.

To eliminate redirections, you should disable the Administration Console’s "Follow changes" feature. To do this, log on to the Administration Console and click Preferences and then Shared Preferences. Clear the Follow Configuration Changes check box and click Save.

To configure HTTP settings for a cluster:

1. If you have not already done so, in the Change Center of the Administration Console, click Lock & Edit.

2. In the left pane of the Console, expand Environment and select Clusters.

3. Select the name of the cluster for which you want to configure HTTP.

4. Select HTTP and enter the following HTTP frontend information. These HTTP settings should be set when host information coming from the URL may be incorrect due to a firewall or proxy.
   - Frontend Host
   - Frontend HTTPPort
   - Frontend HTTPSPort

5. Click Save.

6. To activate these changes, in the Change Center of the Administration Console, click Activate Changes. Not all changes take effect immediately; some require a restart.
Create Customizable Applications

Oracle Metadata Services (MDS) framework allows you to create customizable Oracle Fusion applications. This chapter describes how to configure your application at design time so that it can be customized by end users.

This chapter includes the following sections:

- Introduction to Creating Customizable Applications
- Preparing an Application for Customizations
- Enabling Runtime Customization of Pages and Components

64.1 Introduction to Creating Customizable Applications

With the customization features provided by Oracle Metadata Services (MDS), both developers and customers can customize Oracle Fusion applications. Customizing an application involves taking a generalized application and making modifications to suit the needs of a particular group, such as a specific industry or site.

When an application is customizable, an end user can customize a user interface page at runtime in several ways. For example, the user can use the following methods to customize a page:

- Implicitly: Implicit customizations occur when a customer changes certain component attributes, such as the size of a panel or whether detail content is displayed.

- Using Page Composer: Page Composer enables end users who have the correct permissions, such as administrators, to change a user interface page to suit their company's needs, such as change the page layout, add task flows, documents, and other objects to a page, or wire pages to each other.

- Using the Personalization menu: The Personalization menu in the global area of Oracle Fusion Applications gives an end user access to Page Composer to customize a page to suit their needs. Only the end user sees the changes. End users can also use the Personalization menu to reset a page to display the default content and layout.

- Using the Customization Manager: Customization Manager enables end users who have the correct permissions, such as administrators, to download, upload, reset, and delete application customizations on objects such as pages and task flows.

Some customizations, such as changes to the model or to task flow roles, must be done from Oracle JDeveloper, as described in the "Using Oracle JDeveloper for Customizations" chapter in the Extensibility Guide for Developers. Customizations made from JDeveloper are referred to as design-time customizations. Design-time
customizations that are created and shipped with Oracle Fusion Applications are known as seeded customizations.

**Note:**
End users with the correct permissions can also customize some menus, such as the navigator menu.

For more information about customization, see the "Customizing and Extending Oracle Fusion Applications" chapter in the *Extensibility Guide for Developers*.

A customized application contains a base application and one or more layers of customized metadata content. The customized metadata objects are stored in an MDS repository and, when a customized application is launched, the customized content is retrieved and applied over the base content.

**Note:**
When you run your page in JDeveloper, all customizations created at runtime are, by default, written to a simulated MDS repository directory, which is stored at a temporary location in your system directory. The simulated MDS repository that is configured for the application reflects the metadata information that is contained in the Metadata Archive (MAR).

For more information, see Section 55.2.1, "How to Deploy an Application with Metadata to Integrated WebLogic Server."

To enable runtime customizations of an application, you first prepare the application for customizations, then you enable the runtime customization of pages and components.

### 64.2 Prepare an Application for Customizations

To enable an application for customization, you must complete the following steps in your JDeveloper application workspace in the order shown.

1. Set project properties to enable user and seeded customizations.
2. Configure the persistence change manager parameter to use the composer change manager.
3. Enable end-user translations of customized strings.
4. Add Oracle WebCenter Portal — Composer technology scope to your project. This technology scope contains the components that are used for Page Composer customization.
5. (Optional) Enable user customization of the user interface (UI) shell template.
6. (Optional) Create an IDE connection to your database (in addition to the application connection) to enable access to layer values that are populated from a database call.
After you complete these steps, you can enable runtime customization for the application’s web pages and task flows as described in Enabling Runtime Customization of Pages and Components.

64.2.1 How to Set Project Properties to Enable User and Seeded Customizations

To enable user customizations, you must configure the view project to persist the customized metadata objects to an Oracle Metadata Services (MDS) repository so that the objects are available across sessions. You must also enable seeded customizations so that the page fragments and JSPX pages that you create will be configured to allow customizations.

**Note:**

ADF components (such as controller, model, and business components objects) must have a unique identifier so that they can be customized. ADF components that are generated by JDeveloper are created with identifiers by default, except for fragments and pages in your user interface projects. To cause JDeveloper to generate identifiers for components on pages that you create in your user interface projects, you must explicitly specify this at the project level by enabling seeded customizations.

To set project properties for your view project:

1. In the Application Navigator in JDeveloper, right-click your view project and choose *Project Properties*.

2. In the Project Properties dialog, select *ADF View* to display the ADF View settings.

3. Select *Enable User Customizations* and select *Across Sessions Using MDS*, as shown in Figure 64-1.

4. Select *Enable Seeded Customizations*. 
5. Click OK.

64.2.2 How to Configure the Persistence Change Manager

You must configure the persistence change manager to cause the following runtime behavior to occur:

- The Persist and Don’t Persist attributes that you set for components at design time will govern which implicit changes that the end users make at runtime will be persisted during the session as well as across sessions.

- The changes that end users make in the design view of Page Composer will be stored in the MDS repository.

When you enabled user customizations across sessions by completing the procedure in How to Set Project Properties to Enable User and Seeded Customizations, the IDE added the `CHANGE_PERSISTENCE` context parameter to the view project’s `web.xml` file, and set the parameter to use the filtered persistence change manager. You must modify this parameter to use the composer change manager, and you must add the composer filter and its mapping.

Before you begin:

Modify your view project’s properties to enable user customizations across sessions as described in How to Set Project Properties to Enable User and Seeded Customizations.

To configure the persistence change manager:

1. In the Application Navigator, expand the WEB-INF node for your view project and double-click `web.xml`.

2. In the source editor, change the `org.apache.myfaces.trinidad.CHANGE_PERSISTENCE` context parameter
value to
oracle.adf.view.page.editor.change.ComposerChangeManager, as shown in the following code.

<context-param>
<param-name>org.apache.myfaces.trinidad.CHANGE_PERSISTENCE</param-name>
<param-value>
  oracle.adf.view.page.editor.change.ComposerChangeManager
</param-value>
</context-param>

3. Add the filter and filter-mapping elements for the WebCenterComposerFilter class as shown in bold in Example 64-1.

---

Note:
Filters must be configured in the following order.

a. JpsFilter
b. ApplSessionFilter
c. WebCenterComposerFilter
d. ADFBindingFilter

---

4. Enable sessions for the JSPX pages and task flows that you create in your view project as described in How to Configure Your Project to Use Application User Sessions.

This step, among other modifications, adds the Applications Core and Web Service Data Control libraries to your project, which you need to complete the tasks to prepare your application for customization.

Example 64-1 composerFilter and Mappings in web.xml

....
<!-- composerFilter goes here -->
<filter>
  <filter-name>composerFilter</filter-name>
  <filter-class>
    oracle.adf.view.page.editor.webapp.WebCenterComposerFilter
  </filter-class>
</filter>
<filter>
  <filter-name>adfBindings</filter-name>
  <filter-class>oracle.adf.model.servlet.ADFBindingFilter</filter-class>
</filter>
....

<!-- composerFilter mapping goes here -->
<filter-mapping>
  <filter-name>composerFilter</filter-name>
  <servlet-name>Faces Servlet</servlet-name>
  <dispatcher>FORWARD</dispatcher>
  <dispatcher>REQUEST</dispatcher>
</filter-mapping>
<filter-mapping>
  <filter-name>adfBindings</filter-name>
  <servlet-name>Faces Servlet</servlet-name>
</filter-mapping>
64.2.3 How to Enable Translations of Customized Strings

You must configure your application to enable end users to provide translated values for customized strings at runtime, as described in the "Translating Custom Text" chapter in the Extensibility Guide for Developers.

The resource string editor enables the runtime editing of strings. The changes made using the resource string editor are saved into an application override bundle, which can be translated and imported back into the application.

For more information about configuring the runtime resource string editor, see the "Configuring Runtime Resource String Editing" section in the Developing WebCenter Portal Assets and Custom Components with Oracle JDeveloper.

**Note:**
This procedure only enables resource string editing if the changes are customizations and not user personalizations, as user personalizations do not need to be translated.

To set up runtime resource string editing for customizations:

1. In the Application Navigator in JDeveloper, right-click the application and select Application Properties.
2. In the Application Properties dialog, select Resource Bundles.
3. Click Add.
4. Type the following string in the File Name field in the Select Resource Bundle dialog.

   oracle.adf.view.page.editor.resource.ComposerOverrideBundle.xlf

5. Click Open.
6. Select the Overridden check box.
7. Click OK to save your changes.
8. Open the adf-config.xml file, which is located in the Application Resources > Descriptors > ADF META-INF folder.
9. Add the resource-string-editor element shown in Example 64-2 to the page-editor-config section to enable resource string editing.

**Example 64-2  Configuration to Turn On Resource Editor**

```
<pe:page-editor-config>
  ...
  <resource-string-editor>
    <enabled>
      #{GlobalAreaBackingBean.tipLayerNonUser}
    </enabled>
```
64.2.4 How to Add Composer Technology Scope to Your Project

You must add the Oracle WebCenter Portal — Composer technology scope to access the technologies for consuming Page Composer components and enabling runtime customization.

To add Composer technology scope:

1. Right-click your view project and select Project Properties.
2. Select Technology Scope.
3. Add the Oracle WebCenter Portal — Composer technology scope to your project and click OK.

64.2.5 How to Enable the User Customization of the UI Shell Template

You can configure the UI Shell template so that it can be customizable out-of-the-box. This functionality enables customers to use Page Composer to customize UI Shell pages.

To enable users to customize the UI shell template, you add a link or button to the page fragment from which you want end users to launch the UI Shell template for editing. You then configure the link (or button) to enable end users to modify the template.

To enable the user customization of the UI Shell Template:

1. From JDeveloper, open the JSF page fragment (.jsff) from where the UI Shell can be edited.
2. In the Application Navigator, expand the Data Controls hierarchy to locate and expand the FndUIShellController.
3. Drag and drop the customizeUIShellTemplate operation onto the page, and choose Create > Method > ADF Button or choose Create > Method > ADF Link.
4. In the Edit Action Binding dialog, provide a comma delimited list of fully packaged qualified customization classes for the custClass parameter, as shown in Example 64-3.

Each of the customization classes supplied in the list must be valid and configured in the adf-config.xml file, as shown in Figure 64-2.
If any of the classes cannot be instantiated, or if they are not pre-configured in the \texttt{adf-config.xml} file, an exception is thrown at runtime.

The last customization class specified is the \textit{tip} customization layer and the modifications to the UI Shell is written to this layer. In \texttt{Example 64-3}, the customization of the UI Shell takes place in SiteCC. The purpose of the earlier customization in the list is to view the UI Shell with any other customizations applied.

For information about customization layers, see the "Understanding Customization Layers" section in the \textit{Extensibility Guide for Developers}.

5. Click OK.
6. Open the page definition file for the page fragment.
7. In the editor window, click the \texttt{Source} tab.
8. Add the \texttt{<methodAction>} element shown in \texttt{Example 64-4} to the \texttt{<bindings>} element. The \texttt{id} attribute must be set to custNavigate, which is the key to the \texttt{customizeUIShellTemplate} operation that you dropped on the page in Step 3.
9. Add the view permission for oracle.apps.fnd.appcore.template.customization.TemplateCustomizationUIShellPageDef (the customization page) for each role for which you want to enable the view action as described in \textit{How to Authorize the Runtime Customization of Pages and Task Flows}. To access oracle.apps.fnd.appcore.template.customization.TemplateCustomizationUIShellPageDef in the \texttt{Resource Grants} navigation tab, select \texttt{Web Page} from the \texttt{Resource Type} dropdown list and select \texttt{Show web pages} imported from ADF libraries.

\textit{Example 64-3} \hspace{1em} \textbf{Sample custClass List}

oracle.apps.fnd.appcore.customization.GlobalCC, oracle.apps.fnd.appcore.customization.SiteCC

\textit{Example 64-4} \hspace{1em} \textbf{custNavigate methodAction Binding}

\begin{verbatim}
<bindings>
  <methodAction id="custNavigate" RequiresUpdateModel="true"
              Action="invokeMethod" methodName="navigate"
              IsViewObjectMethod="false" dataControl="FndUIShellController"
              instanceName="FndUIShellController.dataProvider"
              returnName="FndUIShellController.methodResults.navigate_FndUIShellController_dataProvider_naviga"

```
64.2.6 How to Create a Database Connection at the IDE Level

In addition to the application level connection to your application database, you might also need to create an Integrated Development Environment (IDE) connection. The IDE connection is required when implementing design-time customizations from JDeveloper, as described in the "Using Oracle JDeveloper for Customizations" chapter in the Extensibility Guide for Developers.

To create a database connection at the IDE level

1. From the Database Navigator, right-click the ApplicationDB node under your application's node and choose Properties.
2. Make a note of the settings and click Cancel.
3. Right-click IDE Connections and choose New Connection.
4. Enter the settings that you noted in Step 2.
5. Click Test Connection to ensure the settings are correct.
6. Click OK.

The database connection appears under the IDE Connections node.

64.3 Enable Runtime Customization of Pages and Components

After you have completed the non-optional procedures described in Preparing an Application for Customizations, you can enable pages, task flows, and components for runtime customization.

To enable an application's pages, task flows, and components for runtime customization:

1. Enable runtime customization of web pages.
2. Enable end-user personalization for the desired pages.
3. (Optional) Restrict customizations of specific page fragments and components. (By default, customization is allowed for pages, page fragments, and the components on a page.)
4. (Optional) Control the editing of components in Page Composer and control the views that are available in Design view.
5. Authorize runtime customization of pages and task flows.

6. (Optional) Enable and disable persistence of attributes for implicit runtime customizations.

As shown in Figure 64-3, the settings that you make by following the procedures in this section affect whether an end user can customize an object under the following scenarios.

---

**Note:**

For simplicity, the Page Composer scenarios assume that the settings for the page are set to enable the editing of components. For more information, see How to Control the Editing of Components in .

---

- User performs an implicit customizations: If the component can be persisted and customization is allowed, the implicit customization is persisted for the session as well as across sessions.

- User accesses Customization Manager: If the end user has an administrative role or privilege, the user can access the Customization Manager. If the task flow has edit permissions for a user’s role or privilege, the user can import, export, or delete customizations of objects for which customization is allowed.

- User accesses Page Composer to customize a page: If the end user has an administrative role or privilege and customization is allowed for the page (JSPX root), the user can access Page Composer. If the task flow has edit permissions for the user’s role or privilege, the user can customize any object for which customization is allowed.

- User accesses Page Composer to personalize a page: If the page enabled for end-user personalizations and customization is allowed for the page (JSPX root), the user can access the page in Page Composer. If the task flow has edit permissions for the user’s role or privilege, the user can customize any object for which customization is allowed.

- User accesses Page Composer to reset content and layout: If customization is allowed for the page and the task flow has edit permissions for the user’s role or privilege, the user can reset the content and layout for which customization is allowed.

To determine whether customization is allowed, the application looks at the component’s Customization Allowed property to see if it has an explicit value. If there is no explicit value, the application looks at the parent object and continues up the tree, ending at the JSF page fragment (.jspx root), until it finds an explicit value. For example, if Customization Allowed is not set for a component, but is set to true for the page fragment, then customization is allowed for the component. However, if Customization Allowed is false for the component, the application disallows customization of that object, even if the page fragment’s Customization Allowed property is set to true. Conversely, if the page fragment’s Customization Allowed property is set to false, but a component’s property is set to true, the end user can customize the component.

By default the Customization Allowed property for the page (.jspx) root is true. Therefore, you must explicitly set Customization Allowed to false at some level, such as for a component, region, or page (.jspx), to disallow customization for that
object and its child objects. A fragment (.jsff) is loaded by MDS as a separate document and does not inherit the **Customization Allowed** property from the containing page (.jspx).

---

**Note:**

You can optionally use the **Customization Allowed By** property to permit customizations only for certain roles.
64.3.1 How to Enable Pages for Runtime Customization

Perform the following steps to enable runtime customization of a web page using Page Composer:

1. Ensure that the customizable pages have page definition files.
2. Make pages runtime editable by adding Page Composer components to the pages.

3. If you are enabling the runtime addition of content, set up a resource catalog.

4. (Optional) Set up a default catalog definition file to facilitate testing.

64.3.1.1 Ensure Customizable Pages Have Page Definitions

Page definition files define the binding objects that populate data the data in UI components at runtime. A page definition is required for runtime customizations that add additional components such as task flows and portlets. Page definition files can be found under Projects > View Controller > Application Sources > oracle.apps.view. If a required page definition file does not exist, complete the following steps to create one.

To create a page definition file for a JSPX page:

1. In the Application Navigator, right-click the JSPX page and choose Go to Page Definition.

2. If the page does not have a page definition, a Confirm Create New Page Definition dialog appears. Click Yes to create the page.

64.3.1.2 Making a JSPX Document Editable at Runtime

To make a JSPX document editable at runtime, you add Composer components to the page at design time. You use the Panel Customizable component to define an area of the page onto which users can add components at runtime. You use the Layout Customizable component to enable users to lay out its child components in several predefined ways, such as two-column or three-column.

The Layout Customizable and Panel Customizable components are from the Oracle WebCenter Portal — Composer technology scope, which you added when you completed the steps in How to Add Technology Scope to Your Project, and which are available from the Composer page in the Component Palette.

For more information about the Panel Customizable and Layout Customizable components, see the Composer Components section in the Developing WebCenter Portal Assets and Custom Components with Oracle JDeveloper.

64.3.1.3 Setting Up a Resource Catalog

If you have a Panel Customizable component on your page to enable the runtime addition of content, you must set up a resource catalog to list the available content.

To learn how to create a custom resource catalog, see the Creating and Managing Resource Catalogs chapter in the Developing WebCenter Portal Assets and Custom Components with Oracle JDeveloper.

64.3.1.4 Using the Default Catalog Definition File for Testing

A considerable amount of work is involved in setting up a resource catalog. If you want to test runtime customizations before you finish setting up your catalog, you can use the default catalog definition file.

To use the default catalog for testing:
1. Create the application_root/ViewController/src/oracle/adf/rc/metadata directory structure.

2. Copy the Oracle WebCenter Portal default catalog definition file default-catalog.xml to the newly created application_root/ViewController/src/oracle/adf/rc/metadata directory.

Caution:
The Oracle WebCenter Portal default catalog should only be used for testing purposes. You must create your own catalog for production purposes.

64.3.2 How to Enable End-User Personalizations for a Page

Control whether an end user can personalize a page by setting the page's isPersonalizableInComposer property.

Note:
You must enable personalizations for all your dashboards. However, workareas should have personalizations enabled only if absolutely required.

To enable end-user personalizations:

1. In the Application Navigator, select the .jspx page.
2. In the Structure window, select the af:pageTemplate component.
3. In the Property Inspector, select true from the isPersonalizableInComposer dropdown list.

64.3.3 How to Restrict Customization of a Page, Page Fragment, or Component

Customization is enabled for pages, page fragments, and components by default. However, there might be situations where you want to prevent customization for some objects.

You can specify at the page, page fragment, or component level whether customizations for a component are permitted at runtime and who is permitted to customize that component.

Before you begin:
Review the introduction to Enabling Runtime Customization of Pages and Components to understand how the application uses the value of the Customization Allowed property to determine whether an object can be customized.

To restrict customization of a page, page fragment, or component:

1. In the Application Navigator, select the page or fragment for which you want to edit customization properties.
2. If you are restricting the customization of a component, select the component in the Structure window.
3. If the Property Inspector is not open, choose **Property Inspector** from the **View** menu.

4. In the Property Inspector, expand **Customization**.

5. Set the appropriate customization attribute.
   - To disable runtime customization, set **Customization Allowed** to **false**.
   - To restrict customization to specific sets of users and layers, set **Customization Allowed By** to a space-separated list of the security roles for which you wish to permit customization.

For more information about these attributes, see the *Extended Metadata Properties* section in the *Developing Fusion Web Applications with Oracle Application Development Framework*.

For information about customization layers, see the *Understanding Customization Layers* section in the *Extensibility Guide for Developers*.

### 64.3.4 How to Control the Editing of Components in Page Composer

As described in *How to Restrict Customization of a Page_ Page Fragment_ or Component*, all end users who can personalize or customize a page can use Page Composer to change the layout on a page by default.

For some pages, Page Composer enables end users to configure a page in either Source view or Design view. For other pages, end users can only configure the page in Design view. By default, end users can add and edit components in Source view. In Design view the default is to allow end users to add components to a page but not edit existing components. You can configure the page to allow users to select components for editing in Design view, as shown in *Figure 64-4*, or you can configure the page to disallow both adding and editing components.

*Figure 64-4 Page Configured to Allow Users to Add and Edit Components*

If a page has properties with EL expressions that you do not want changed, you can set all existing property EL expressions to read-only. For example, the page in *Figure 64-4* was configured to disallow the editing of existing EL expressions and, therefore, the user cannot edit the **Action Listener** property. In addition, you can configure a page to disallow entering new EL expressions. That is, you can prevent users from entering EL expressions in property fields that do not currently contain EL expressions.
Table 64-1 lists the properties that you set at the af:pageTemplate level to control the editing of components in Page Composer.

### Table 64-1  af:pageTemplate Properties for Page Composer

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Supports EL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowELInComposer</td>
<td>Boolean</td>
<td>No</td>
<td>Specifies if EL expressions can be entered for form element properties. The default value is true.</td>
</tr>
<tr>
<td>designViewsInComposer</td>
<td>String</td>
<td>No</td>
<td>Specifies the views available to the user in Page Composer Design view. This property can take a space-separated list of elements. The following options are available:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• all: Makes both Add Content and Select views available and displays a toolbar with buttons to switch between views.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• add-content (default): Makes the Add Content view the only available view.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• select add-content: The same as the all option.</td>
</tr>
<tr>
<td>protectELInComposer</td>
<td>Boolean</td>
<td>No</td>
<td>Specifies if existing EL expressions in the page's form elements are read-only. The default value is false.</td>
</tr>
</tbody>
</table>

Table 64-2 lists all possible combinations of allowELInComposer and protectELInComposer property values and describes how each combination affects the user's ability to add and edit EL expressions.

### Table 64-2  Affects of allowELInComposer and protectELInComposer Property Settings

<table>
<thead>
<tr>
<th>Combination</th>
<th>Affect</th>
</tr>
</thead>
<tbody>
<tr>
<td>protectELInComposer = true, allowELInComposer = true</td>
<td>Users can add an EL expression to a property if it does not currently have an EL expression and if EL expressions are allowed for that property. However, users cannot edit existing EL expressions.</td>
</tr>
<tr>
<td>protectELInComposer = false, allowELInComposer = true</td>
<td>Users can add and edit EL expressions in properties for which EL expressions are allowed</td>
</tr>
</tbody>
</table>
Table 64-2  (Cont.) Affects of allowELInComposer and protectELInComposer
Property Settings

<table>
<thead>
<tr>
<th>Combination</th>
<th>Affect</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>protectELInComposer = false</code></td>
<td>Users cannot add an EL expression to a property if it does not currently contain an EL expression, but they can edit existing EL expressions.</td>
</tr>
<tr>
<td><code>allowELInComposer = false</code></td>
<td></td>
</tr>
<tr>
<td><code>protectELInComposer = true</code></td>
<td>User cannot add or edit EL expressions.</td>
</tr>
<tr>
<td><code>allowELInComposer = false</code></td>
<td></td>
</tr>
</tbody>
</table>

64.3.5 How to Authorize the Runtime Customization of Pages and Task Flows

As illustrated in Figure 64-3 a user can edit customizable components in a user interface page at runtime only if they have permission to edit the page and permission to edit the task flow that contains the component. For example, an end user can only customize components in a task flow using Page Composer if that user has permission to customize the task flow. You use the `jazn-data.xml` file to define which roles can edit the page or task flow.

**Before you begin:**

Enable the desired pages for editing in Page Composer as described in Enabling Runtime Customization of Pages and Components.

**To authorize pages and task flows for runtime customizations:**

1. In the Application Resources panel, expand Descriptors, expand META-INF nodes, and then double-click `jazn-data.xml`.

   **Tip:**

   If the `jazn-data.xml` file does not exist, you can create it by right-clicking the META-INF node, selecting New Oracle Deployment Descriptor, selecting `jazn-data.xml`, and then clicking Finish.

2. In the overview editor, click the Resource Grants navigation tab, as shown in Figure 64-6.
3. Select **Web Page** from the **Resource Type** dropdown list.

   You use the **Web Page** resource type for both web pages and page fragments.

4. Set the **Source Project** to **ViewController**.

5. For each Page Composer enabled web page, select the **Customize** action for each role for which you want to enable page customization.

   You must also authorize customization for the task flows, as described in the following two steps.

6. Select **Task Flow** from the **Resource Type** dropdown list.

7. For each task flow, select the **Customize** action for each role for which you want to enable customization of the components in the task flow.

8. If you want the task lists that are exposed in your pages to be customizable for your application, select the entry for **TaskList** in the **Resources** list, and, for each role for which you want to enable task list customization, select the **customize** and **grant** actions.

   For more information, see the **Implementing Task Flow Security** section in the Developing WebCenter Portal Assets and Custom Components with Oracle JDeveloper.

### 64.3.6 How to Persist Implicit Runtime Customizations

Certain ADF Faces components have attributes that can be saved during a user session. For example, if a user expands a panel box component, the box will be expanded when the user returns to the page. This type of change is referred to as implicit runtime customization. For information about which component properties can be persisted, see the **Allowing User Customizations at Runtime** chapter in the Developing Fusion Web Applications with Oracle Application Development Framework.
In How to Set Project Properties to Enable User and Seeded Customizations, you configured the application to persist implicit runtime customizations across sessions as well as within sessions.

The attributes that can be persisted are set in the tag library, but you can override these settings. For example, you might not want the end users to change column widths, but you want all other default attribute changes for columns to be persisted. You set and unset these values in the Overview Editor for the `adf-config.xml` file, as described in the Configuring User Customizations section in the Developing Fusion Web Applications with Oracle Application Development Framework. You can also override these settings for specific components, as described in the Controlling User Customizations in Individual JSF Pages section in the Developing Fusion Web Applications with Oracle Application Development Framework.
This chapter explains how to use extensions to Oracle Enterprise Scheduler to manage job request submissions in the context of Oracle Fusion Applications. This chapter includes the following sections:

- Introduction to Extensions
- Understanding Standards and Guidelines
- About Creating and Implementing a Scheduled Job in
- About Creating a Job Definition
- About Configuring a Spawned Job Environment
- About Implementing a PL/SQL Scheduled Job
- About Implementing a SQL*Plus Scheduled Job
- About Implementing a SQL*Loader Scheduled Job
- About Implementing a Perl Scheduled Job
- About Implementing a C Scheduled Job
- About Implementing a Host Script Scheduled Job
- About Implementing a Java Scheduled Job
- About Elevating Access Privileges for a Scheduled Job
- About Creating an User Interface for Submitting Job Requests
- About Submitting Job Requests Using the Request Submission API
- About Defining Postprocessing Actions for a Scheduled Job
- About Monitoring Scheduled Job Requests Using an UI
- About Using a Task Flow Template for Submitting Scheduled Requests Through an UI
- About Securing UIs
- About Integrating Scheduled Job Logging with
- About Logging Scheduled Jobs and Writing to Output Files
65.1 Introduction to Oracle Enterprise Scheduler Extensions

Oracle Enterprise Scheduler provides the ability to run different job types, including Java, PL/SQL, and spawned jobs. Jobs can run on demand, or be scheduled to run in the future.

Oracle Enterprise Scheduler provides scheduling services for the following purposes:

- Distributing job request processing across a grid of application servers.
- Running Java, PL/SQL, and process or spawned jobs.
- Processing multiple jobs concurrently.
- Running the same job in different languages.

Using Oracle JDeveloper, application developers can create and implement jobs. While implemented in JDeveloper, Oracle Enterprise Scheduler runs the jobs. APIs provide an interface between jobs executed within applications developed in JDeveloper and Oracle Enterprise Scheduler.

The Oracle JDeveloper extensions to Oracle Enterprise Scheduler enable the following:

- Running scheduled Oracle Business Intelligence Publisher (Oracle BI Publisher), spawned, Java, PL/SQL, Perl, SQL*Plus, SQL*Loader, and C jobs.
- Running the same job in multiple locales, time zones, currencies, and so on.
- Creating log and output files for jobs, as well as acting upon those files, such as enabling notifications.
- Creating Oracle Application Development Framework (Oracle ADF) task flows to schedule jobs and job sets, as well as monitor job requests.

Before you begin:

Install Oracle Enterprise Scheduler to Oracle WebLogic Server. For more information, see Chapter 2, "Setting Up Your Development Environment."

65.2 Understanding Standards and Guidelines

The following standards and guidelines apply to working with extensions to Oracle Enterprise Scheduler:

- Always use the preconfigured job types provided when defining metadata for job definitions.

65.3 About Creating and Implementing a Scheduled Job in Oracle JDeveloper

Submitting job requests from an Oracle Fusion application requires developing the following components:

- A job definition, created in JDeveloper
- The Java, PL/SQL, SQL*Loader, SQL*Plus, Perl, C, or host scripts job implementation
• A user interface enabling end users to submit job requests and/or additional properties for the job

A wizard enables defining a new job within the context of an Oracle Fusion application. The job can be any one of the following types: Java, PL/SQL, SQL*Loader, SQL*Plus, Perl, C, or host scripts.

65.3.1 How to Create and Implement a Scheduled Job in JDeveloper

Creating and implementing a scheduled job in JDeveloper involves creating a package or class from which to call the job, as well as defining a job definition. The job must then be deployed and tested, and a job request submission interface defined.

To create and implement a scheduled job in JDeveloper, do the following:

1. Create a package, class, or job, and include the minimum required methods or functions.
   - Define the job request
   - Define any subrequests, if required.

2. If a job requires parameters to be filled in by end users using an Oracle ADF user interface, define a standard ADF Business Components view object with validation.

   For example, if a job requires information regarding duration, date, and time, create an ADF Business Components view object with the properties duration, date, and time.

3. Create a job definition in JDeveloper using the wizard.

   If using an ADF Business Components view object to collect additional values at runtime from end users, specify the name of the view object as a property of the job definition.

4. Deploy the job.

5. Test the job.

6. Create the end user job request submission interface.

   For more information about creating the end user job request submission interface, see About Creating an User Interface for Submitting Job Requests.

65.3.2 What Happens When You Create and Implement a Scheduled Job in JDeveloper

Once a job has been defined, it is implemented in Java, PL/SQL, SQL*Loader, SQL*Plus, Perl, C or a host script. The Fusion application requests Oracle Enterprise Scheduler to run the implemented jobs using the provided APIs.

65.3.3 What Happens at Runtime: How a Scheduled Job Is Created and Implemented in JDeveloper

An Oracle ADF interface is provided to enable application end users to submit job requests from an Oracle Fusion application. The Oracle ADF interface is integrated into an Oracle Fusion application. As soon as a job request is submitted through the interface, Oracle Enterprise Scheduler runs the job as scheduled.
65.4 About Creating a Job Definition

To submit a job request, you must first create a job definition.

Related Links

The following documents provide additional information related to subjects discussed in this section:

- For more information about defining an Oracle BI Publisher job, see the Oracle Fusion Middleware Report Designer’s Guide for Oracle Business Intelligence Publisher, Oracle Fusion Middleware Administrator’s Guide for Oracle Business Intelligence Publisher, and the Oracle Fusion Developer’s Guide for Oracle Business Intelligence Publisher.

- For more information about using entity objects, see the section Creating a Business Domain Layer Using Entity Objects in Developing Fusion Web Applications with Oracle Application Development Framework guide.

- For more information about log levels, see the Developing Applications for Oracle Enterprise Scheduler guide.

65.4.1 Understanding Job Definitions and Job Types

A job definition and job type are required to submit a job request.

- **Job Definition**: This is the basic unit of work that defines a job request in Oracle Enterprise Scheduler.

- **Job Type**: This specifies an execution type and defines a common set of properties for a job request.

The extensions to Oracle Enterprise Scheduler provide the following execution types:

- **JavaType**: for job definitions that are implemented in Java and run in the container.

- **SQLType**: for job definitions that run as PL/SQL stored procedures in a database server.

- **CJobType**: for job definitions that are implemented in C and run in the container.

- **PerlJobType**: for job definitions that are implemented in Perl and run in the container.

- **SqlLdrJobType**: for job definitions that are implemented in SQL*Loader and run in the container.

- **SqlPlusJobType**: for job definitions that are implemented in SQL*Plus and run in the container.

- **BIPJobType**: for job definitions that are executed as Oracle BI Publisher reports. Oracle BI Publisher jobs require configuring the parameter `reportID`.

- **HostJobType**: for job definitions that run as host scripts executed from the command line.
65.4.1.1 How to Create a Job Definition in Oracle JDeveloper

If your job definition requires additional properties to be filled in by end users at submission time, you’ll need to create a view object that defines these properties. The view object must be associated with the job definition you create. The view object is later associated with the user interface you create to allow end users to submit job requests along with the properties at submission time.

For more information about defining properties to be filled in at runtime by end users, see About Creating a User Interface for Submitting Job Requests.

To create a new job definition in JDeveloper, do the following:

1. In Oracle JDeveloper, create an Oracle Fusion web application by clicking the Application Menu icon on the Application Navigator, selecting New Project > Projects > Generic Project and clicking OK.

2. Right-click the project and select Properties. In the Resources tab, add the directory $MW_HOME/jdeveloper/integration/ess/extJobTypes.

3. If your job includes any properties to be filled in by end users using an Oracle ADF user interface at runtime, create an ADF Business Components view object with validation and the parameters to be filled in by end users.

   a. Right-click the Model project and select Properties. In the Resource Bundle section, configure one bundle per file and select resource bundle type Xliff Resource Bundle.

   b. Define attributes for the view objects sequentially, ATTRIBUTE1, ATTRIBUTE2, and so on, with an attribute for each required parameter. Use ADF Business Components attribute control hints to specify required prompt, validation, and formatting for each parameter.

   c. Add the property parametersVO to your job definition and specify the fully qualified path of the view object as the value of parametersVO. For example, set parametersVO to oracle.my.package.TestVO. A maximum of 100 attributes can be used for parametersVO. The attributes should be named incrementally, for example ATTRIBUTE1, ATTRIBUTE2, and so on.

   d. Define the following required properties:

      • jobDefinitionName: The short name of the job.
      • jobDefinitionApplication: The short name of the application running the job.
      • jobPackageName: The name of the package running the job.

Additional properties can be defined as shown in Table 65-1.
### Table 65-1  Additional Job Definition Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>completionText</td>
<td>An optional string value that can be used to communicate details of the final state of the job. This property value is displayed in the UI used to monitor job request submissions in the details section of the job request. It can be useful for displaying a short explanation as to why a request ended in an error or warning state.</td>
</tr>
<tr>
<td>CustomDatacontrol</td>
<td>The name of the data control for the application to which the parameter task flow is bound. Following is an example.</td>
</tr>
<tr>
<td></td>
<td>&lt;parameter name=&quot;CustomDatacontrol&quot; data-type=&quot;string&quot;&gt;ExtParameterAM&lt;/parameter&gt;</td>
</tr>
<tr>
<td></td>
<td>Use this property when adding a custom task flow to an Oracle ADF user interface used to submit job requests at runtime. For more information, see How to Add a Custom Task Flow to an User Interface for Submitting Job Requests.</td>
</tr>
<tr>
<td>defaultOutputExtension</td>
<td>The suffix of the output file. Possible values are txt, xml, pdf, html.</td>
</tr>
<tr>
<td>enableTimeStatistics</td>
<td>A Boolean parameter that enables or disables the accumulation of time statistics (Y or N).</td>
</tr>
<tr>
<td>enableTrace</td>
<td>A numerical value that indicates the level of tracing control for the job. Possible values are as follows:</td>
</tr>
<tr>
<td></td>
<td>• 1: Database trace</td>
</tr>
<tr>
<td></td>
<td>• 5: Database trace with bind</td>
</tr>
<tr>
<td></td>
<td>• 9: Database trace with wait</td>
</tr>
<tr>
<td></td>
<td>• 13: Database trace with bind and wait</td>
</tr>
<tr>
<td></td>
<td>• 16: PL/SQL profile</td>
</tr>
<tr>
<td></td>
<td>• 17: Database trace and PL/SQL profile</td>
</tr>
<tr>
<td></td>
<td>• 21: Database trace with bind and PL/SQL profile</td>
</tr>
<tr>
<td></td>
<td>• 25: Database trace with wait and PL/SQL profile</td>
</tr>
<tr>
<td></td>
<td>• 29: Database trace with bind, wait and PL/SQL profile</td>
</tr>
<tr>
<td>executionLanguage</td>
<td>Stores the preferred language in which the job request should run.</td>
</tr>
<tr>
<td>executionNumchar</td>
<td>The numeric characters used in the preferred language in which the job runs, as defined by executionLanguage.</td>
</tr>
<tr>
<td>executionTerritory</td>
<td>The territory of the preferred language in which the job runs, as defined by executionLanguage.</td>
</tr>
<tr>
<td>EXT_PortletContainerWeb</td>
<td>Specifies the name of the web module for the Oracle Enterprise Scheduler UI application to use as a portlet when submitting a job request. The Oracle Enterprise Scheduler central UI looks up the producer from the topology based on the registered producer application name derived from EXT_PortletContainerWebModule.</td>
</tr>
</tbody>
</table>

---

About Creating a Job Definition

---

65-6  Oracle Fusion Applications Developer's Guide
Table 65-1  (Cont.) Additional Job Definition Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incrementProc</td>
<td>Enables a PL/SQL procedure evaluated at runtime which calculates the next set of date parameter values for a recurring request. Enter the name of the PL/SQL procedure. The procedure expects one argument—a number signifying the change in milliseconds between the start dates of the first and current requests.</td>
</tr>
</tbody>
</table>

```sql
-- incr_test - Sample PL/SQL incrementProc procedure
-- This procedure gets the list of arguments to be incremented
-- using the incrementProcArgs property and increments each
-- argument by the delta provided. This behavior is identical
-- to the default behavior if no incrementProc is set for the
-- job.

procedure incr_test(   delta IN number ) is
    request_id number;
    incrProcArgs varchar2(200);
    curr_arg_n varchar2(100);
    curr_arg_v varchar2(2000);
    del_pos number := 0;
    prev_pos number := 1;
    old_date date;
    new_date date;
    delta_days number;
    begin
        request_id := FND_JOB.REQUEST_ID;
        delta_days := delta / (1000*60*60*24);

        -- incrProcArgs must be defined for this procedure to be
        -- called.
        incrProcArgs := ESS_RUNTIME.GET_REQPROP_VARCHAR(request_id,
                                                         FND_JOB.INCR_PROC_ARGS_P) || ',

        LOOP
            del_pos := INSTR(incrProcArgs, ',', prev_pos);
            EXIT WHEN del_pos = 0;
            curr_arg_n := FND_JOB.SUBMIT_ARG_PREF_P || SUBSTR(incrProcArgs,
                                                             prev_pos, del_pos-prev_pos);
            curr_arg_v := ESS_RUNTIME.GET_REQPROP_VARCHAR(request_id,
                                                            curr_arg_n);

            old_date := FND_DATE.CANONICAL_TO_DATE(curr_arg_v);
            new_date := old_date + delta_days;
            ESS_RUNTIME.UPDATE_REQPROP_VARCHAR(request_id, curr_arg_n,
                                                 FND_DATE.DATE_TO_CANONICAL(new_date));

            prev_pos := del_pos+1;
        END LOOP;
    end incr_test;
```

About Creating a Job Definition

Work with Extensions to Oracle Enterprise Scheduler  65-7
### Table 65-1 (Cont.) Additional Job Definition Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incrementProcArgs</td>
<td>A list of comma-separated date arguments to be incremented. The incrementProc property is used to increment these values. Alternatively, a default calculation is used if the property incrementProc is not defined. Enter a list of argument numbers to identify which job arguments are to be incremented (for example, &quot;1, 2, 5&quot;). In the incrementProc example shown above, an incrementProc procedure calculates the next set of date parameter values for a recurring request. The procedure expects one argument: a number signifying the change in milliseconds between the start dates of the first and current requests.</td>
</tr>
<tr>
<td>logLevel</td>
<td>The level at which events are logged (between 0 and 4). Each job type has a logLevel of 1 by default. This optional value is used to override the job type logLevel in the job definition.</td>
</tr>
<tr>
<td>optimizerMode</td>
<td>This flag enables setting the database optimizer mode for the job. Optimizer mode is useful for fine-tuning performance.</td>
</tr>
<tr>
<td>parametersVO</td>
<td>The ADF Business Components view object you define for additional properties to be entered at runtime by end users using an Oracle ADF user interface.</td>
</tr>
<tr>
<td>ParameterTaskflow</td>
<td>Enter the name of the task flow as a parameter. The name of the taskflow.xml file must be the same as the taskflowId. Following is an example.</td>
</tr>
<tr>
<td></td>
<td><code>&lt;parameter name=&quot;ParameterTaskflow&quot;  data-type=&quot;string&quot;&gt;/WEB-INF/oracle/apps/prod/project/ParamTestTaskFlow.xml#ParamTestTaskFlow&lt;/parameter&gt;</code></td>
</tr>
<tr>
<td></td>
<td>Use this property when adding a custom task flow to an Oracle ADF user interface used to submit job requests at runtime. For more information, see How to Add a Custom Task Flow to an User Interface for Submitting Job Requests.</td>
</tr>
<tr>
<td>reportID</td>
<td>The Oracle BI Publisher report value specified in the Oracle BI Publisher repository. Required parameter for Oracle BI Publisher jobs only.</td>
</tr>
<tr>
<td>rollbackSegment</td>
<td>Enables setting a database rollback segment for the job, which will be used until the first commit. When implementing the rollback segment, use FND_JOB.AF_COMMIT and FND_JOB.AF_ROLLBACK to commit and rollback.</td>
</tr>
<tr>
<td>srsFlag</td>
<td>A Boolean parameter (Y or N) that controls whether the job displays in the job request submission user interface (see About Creating an User Interface for Submitting Job Requests).</td>
</tr>
<tr>
<td>SYS_runasApplicationID</td>
<td>Enables elevating access privileges for completing a scheduled job. For more information about elevating access privileges for the completion of a particular job, see About Elevating Access Privileges for a Scheduled Job.</td>
</tr>
</tbody>
</table>

4. Create a new job. From the New Gallery, select **Business Tier > Enterprise Scheduler Metadata** and click **Job Definition**.
5. In the Job Definition Name & Location page in the Job Definition Creation wizard, do the following:

- **Name**: Enter a name for the job.
- **JobType**: Select the job type from the drop-down list.

Click **Finish**. The new job definition displays.

6. Edit the following properties in the job definition as required for the selected job type:

- **JavaJobType**: Uncheck the read-only checkbox next to `className` and set its value to the value of the business logic class.

- **PlsqlJobType**: Uncheck the read-only checkbox next to `procedureName` and set its value to the name of the procedure (such as `myprocedure.proc`). Create a new parameter named `numberOfArgs`. Set `numberOfArgs` to the number of job submission arguments, excluding `errbuf` and `retcode`.

- **CJobType**: Add the parameter `executableName` and set its value to the name of the C job to be executed. The executable file identified by the `executableName` parameter must exist in the directory `$APPLICATIONS_BASE/$APPLBIN`.

- **PerlJobType**: Add the parameter `executableName` and set its value to the name of the Perl script.

- **SqlLdrJobType**: Add the parameter `executableName` and set its value to the name of the control file to be executed (located under `PRODUCT_TOP/$APPLBIN`). Add SQL*Loader options such (such as `direct=yes`) as a `sqlldr.directoption` parameter in the job definition.

- **SqlPlusJobType**: Add the parameter `executableName` and set its value to the name of the SQL*Plus job script to be executed (located under `PRODUCT_TOP/$APPLSQL`).

- **HostJobType**: Add the parameter `executableName` and set its value to the name of the host script job to be executed. The executable file identified by the `executableName` parameter must exist in the directory `PRODUCT_TOP/$APPLBIN`.

**Note:**

Configure the `$APPLBIN` and `$APPLSQL` variables in the `environment.properties` file. The `$APPLBIN` and `$APPLSQL` variables point to the location of executable files under `PRODUCT_TOP`. These variables enable the extensions to Oracle Enterprise Scheduler to locate the jobs to be run. Typically, these variables are set in a preexisting environment properties file in the system.

---

65.4.2 How to Define File Groups for a Job

A file group is a collection of output files such as text files, XML files, and so on. File groups enable categorizing files together for a specific purpose, such as file groups for human resources or financial reports.
File groups are used for postprocessing jobs such as Business Intelligence Publisher jobs. Using postprocessing actions, the results of a job can be saved as an HTML file, for example, or printed. File groups specify the type of postprocessing action to be taken for a given job.

There are two types of file groups: output and layout. Postprocessing layout actions create additional output files using the job request output files. For example, an XML job output file can be processed as an HTML or PDF file.

Postprocessing output actions act upon job request output files by printing, faxing, or emailing the files, for example. Output postprocessing actions can be taken on job request output files, as well as files created by layout postprocessing actions. For example, a job request output XML file can be converted to a PDF file using layout postprocessing actions, and then emailed using output postprocessing actions.

To define file group properties, do the following:

1. In the job definition for which you want to define postprocessing, define a file group.
   a. Name the property Program.FMG.
   b. For the value of the property, enter a list of comma-separated File Management Groups, where each file group is prefixed by an L or O to indicate a layout or output file group, respectively. A sample file group property is shown in Example 65-1.

   Three file groups are listed in this example.

2. In the job definition, create a property containing a regular expression used to filter the files in the output work directory of the job request. Any output files that match the filter will be part of the relevant file group.

   Example regular expressions are shown in Example 65-2, Example 65-3, and Example 65-4.

   An example of file group properties in a job definition is shown in Example 65-5. These properties specify the use of the Business Intelligence Publisher postprocessing action on the MYXML file group, followed by the print postprocessing action on either ALL or PDF file groups.

3. Optionally, rename the file group and store it in the Oracle Metadata Service repository so that it displays in a more user-friendly way in the scheduled job request submission UI.

   Example 65-1  File Group Property Sample Value
   Program.FMG = L.MYXML, O.ALL, O.PDF

   Example 65-2  File Group Regular Expression Filtering for All Files with the Suffix XML
   MYXML = '.*.\xml$'

   Example 65-3  File Group Regular Expression Filtering for All Files
   ALL = '.*$'

   Example 65-4  File Group Regular Expression Filtering for All Files with the Suffix PDF
   PDF = '.*.\pdf$'
Example 65-5  File Group Properties with File Group Regular Expression Filtering

Program.FMG = L.MYXML, O.ALL, O.PDF
MYXML = '.*\.xml$'  ALL = '.*$'  PDF = '.*\.pdf$'

65.4.3 What Happens When You Create a Job Definition

The job definition is written to an XML file called <job name>.xml.

65.4.4 What Happens at Runtime: How Job Definitions Are Created

The Oracle Fusion application passes the job definition file to Oracle Enterprise Scheduler, which runs the job defined in the file.

65.5 About Configuring a Spawned Job Environment

Configuring a spawned job involves creating an environment file and configuring an Oracle wallet.

65.5.1 About Creating an Environment File for Spawned Jobs

Spawned jobs require an environment.properties file to provide the correct environment for execution. The environment.properties file should be located in the config/fmwconfig directory under the domain.

Additional environment variables may be added to the same directory in a similar file called env.custom.properties. Variables defined in this file take precedence over those in the environment.properties file.

Similarly, server-specific environment variables may be set in the server config directory in files called environment.properties and env.custom.properties.

65.5.1.1 How to Create an Environment File for Spawned Jobs

The following variables are used to identify the correct interpreters for various spawned job types:

- **AFSQLPLUS**: The executable for SQL*Plus scripts.
- **AFSQLLDR**: The executable for SQL*Loader uploads.
- **AFPERL**: The Perl interpreter.
- **ATGPF_TOP**: The TOP directory for ATGPF files, needed to locate key files for SQL*Plus and Perl jobs.

The following environment properties are available to all spawned jobs:

- **REQUESTID**: The request ID of the current job request.
- **WORK_DIR_ROOT**: The directory on the local file system where the request can perform file operations.
- **OUTPUT_WORK_DIR**: The directory to which the job writes all output files.
- **LOG_WORK_DIR**: The directory to which the job writes all log files.
- **INPUT_WORK_DIR**: The directory to which input files are saved before the job is spawned.
• OUTFILE_NAME: The default name for the job output file.

• LOGFILE_NAME: The name of the log file for the job.

• USER_NAME: The name of the user submitting the job. The job runs in the context of this user.

• REQUEST_HANDLE: The Oracle Enterprise Scheduler request handle for the current request.

The environment variables must point to the client ORACLE_HOME and environment so that spawned jobs can connect to the database.

**Note:**

Ensure the variables you define in the environment.properties file do not include any trailing spaces. Follow the guidelines required by java.util.properties.

Restart the server after editing the environment.properties file.

To create an environment file, do the following:

1. Use a text editor to create an environment.properties file for the spawned job.

2. Set the following environment variables in the environment.properties file:
   - LD_LIBRARY_PATH
   - ORACLE_HOME
   - PATH: The full path of the spawned job. In Windows environments, the PATH must include all directories that are typically part of LD_LIBRARY_PATH.
     - TNS_ADMIN: The directory which stores files related to the database connection (such as tnsnames.ora, sqlnet.ora).
   - TWO_TASK: The TNS name identifying the database to which spawned jobs should connect. In Windows environments, the environment variable is LOCAL.

3. Configure the following variables, which are required to locate spawned jobs:
   - APPLBIN: C executables and SQL*Loader control files must reside in the $APPLBIN directory under the product TOP.
   - APPL_TOP: Set this property to the top level directory where the bin directory of C executables resides.
   - APPLSQL: SQL*Plus scripts must reside in the $APPLSQL directory under the product TOP. This means that the product TOP should be accessible to the environment.
   - ATGPF_TOP: This variable is required for SQL*Plus jobs. This should point to where the wrapper script is available.

4. Save the environment.properties file and restart the server.
65.5.2 How to Configure an Oracle Wallet for Spawned Jobs

Use the TNS_ADMIN and ORACLE_HOME variables specified in the environment.properties file created in About Creating an Environment File for Spawned Jobs.

A configured Oracle wallet enables spawned jobs to connect to the database at the command line. A provisioned Oracle Fusion Applications environment will have this wallet preconfigured.

To configure an Oracle wallet for the spawned job, do the following:

1. At the prompt, enter the following commands as shown in Example 65-6.

2. When prompted, choose a password for the wallet.

3. At the prompt, enter the following command as shown in Example 65-7.

   where TWO_TASK is the variable in the environment.properties file and <fusion password> is the password for the fusion user name.

   This command creates permissions for accessing the wallet.

4. When prompted, enter the wallet password created earlier.

5. In a text editor, create a file called sqlnet.ora that includes the lines shown in Example 65-8.

6. In a text editor, create a file called tnsnames.ora that includes the lines shown in Example 65-9.

7. [Only for Unix OS] Execute the following commands as shown in Example 65-10.

   The first command enables anyone to read and execute files in the directory, while reserving write access to the directory creator.
   The second command enables only the file owner to read, write and execute the file, while anyone can read the file.

8. Test the wallet by connecting to it. Execute the following command as shown in Example 65-11.

Example 65-6 Creating a Wallet

If you are using a Linux operating system, use these commands:

   cd $TNS_ADMIN
   mkdir wallet
   mkstore -wrl ./wallet -create

If you are using a Windows operating system, use these commands:

   cd %TNS_ADMIN%
   mkdir wallet
   mkstore -wrl wallet -create

Example 65-7 Creating Wallet Credentials

If you are using a Linux operating system, use this command:

   mkstore -wrl ./wallet -createCredential <$TWO_TASK> fusion_runtime <fusion_runtime_password password>

Work with Extensions to Oracle Enterprise Scheduler 65-13
If you are using a Windows operating system, use this command:

```
mkstore -wl wallet -createCredential <%TWO_TASK%> fusion_runtime
<fusion_runtime_password password>
```

**Example 65-8  Create a File Called sqlnet.ora**

If you are using a Linux operating system, use these commands:

```sql
SQLNET.WALLET_OVERRIDE = TRUE
WALLET_LOCATION =
  (SOURCE =
   (METHOD = FILE)
   (METHOD_DATA =
    (DIRECTORY = <$TNS_ADMIN>/wallet)
  )
)
```

If you are using a Windows operating system, use these commands:

```sql
SQLNET.WALLET_OVERRIDE = TRUE
WALLET_LOCATION =
  (SOURCE =
   (METHOD = FILE)
   (METHOD_DATA =
    (DIRECTORY = <%TNS_ADMIN%>/wallet)
  )
)
```

**Example 65-9  Create a File Called tnsnames.ora**

```
dbnname =
  (DESCRIPTION =
   (ADDRESS =
    (PROTOCOL = TCP)
    (HOST = host.example.com)
    (PORT = 1521)
   )
   (CONNECT_DATA = (SID=sidname))
  )
```

**Example 65-10  Set Directory and File Permissions**

```
chmod 755 wallet
chmod 744 wallet/cwallet.sso
```

**Example 65-11  Connect to the Wallet**

If you are using a Linux operating system, use this command:

```
sqlplus /@$TWO_TASK$
```

If you are using a Windows operating system, use this command:

```
sqlplus /@<%TWO_TASK%>
```

### 65.5.3 About Migrating Spawned Job Environment Properties to the Production Environment

Oracle Enterprise Scheduler provides a plug-in to migrate spawned job environment properties from the testing environment to the production environment. The
The migration plug-in moves the data-sources, LDAP, JMS, and other configurations that are part of the domain of the testing environment to the production environment.

The plug-in aims to incorporate properties created for spawned job processing. As such, the test to production plug-in parses the environment.properties file created in the testing environment and migrates it to the production environment. The file resides in the location specified in the testing environment as defined in the system property ess.config.dir.

As the properties to be moved reside in a flat file, it is unnecessary to annotate any MBeans with the property MovableProperty. Changes in the Oracle Enterprise Scheduler connections.xml are handled by the plug-in.

The migration plug-in includes the following components: CopyConfig, MovePlans, PasteConfig.

- **CopyConfig**: This script accomplishes the following tasks.
  - Reads the EssConfigDir location property from the system.
  - Reads and parses the environment.properties file.
  - Creates a MovableComponent with the name ESS-EXT and creates another MovableComponent with the name ENV-PROPS which it then adds to the ESS-EXT movable property.
  - Loops through every property in the environment.properties file and adds it to the ESS-EXT MovableComponent as a ConfigProperty.
  - Moves all the properties available in the properties file of the testing environment to the same file of the production environment.
  - Analyzes properties to determine whether they are changeable in the production environment. If they are, then those properties are defined as READ_WRITE. If they are not changeable, the properties are defined as READ_ONLY.
  - Returns the list of MovableComponent objects.
  - In post-processing, sets the EssConfigDir property to componentProperties so that the PasteConfig script can fetch the correct value.
  - Generates the MovePlan.xml file, whose values can be modified as required for the production environment.

- **MovePlans**: The environment.properties file is located outside the domain home or Oracle Fusion Middleware home. As such, a test to production plug-in must extract these properties and makes them movable through the test to production framework. The Plug-in extracts the key and value of the properties defined in the environment.properties file and makes them available in MovePlan.xml so that the values may be modified accordingly to suit the needs of the production environment.

  An example of the hierarchy of a MovePlans.xml file is shown in Example 65-12.

- **PasteConfig**: This script writes all the required values to the environment.properties file of the production environment.
– Gets the MovableComponent from FMWT2PPasteBean with type ESS-EXT. Gets the internal MovableComponent with name ENV-PROPS from ESS-EXT.

– Creates a new environment.properties file in the UserFileDir location of the production environment.

– Extracts the ConfigProperty values added to this MovableComponent and constructs an output steam.

– Writes all the values to the production environment.properties file.

**Example 65-12  MovePlan.xml File**

```xml
movePlan

  |_movableComponent (componentType:J2EEDomain, componentName:base_domain)
    //Default root movable component
  |_movableComponent (componentType:ESS-EXT, componentName:'Oracle Enterprise Scheduler Extension components')
    |_moveDescriptor
      |_configGroup (type: ENVIRONMENT_PROPERTIES)
        |_configProperty (id="/machine/user/instance/ess/config")
          // Location of the environment.properties
        |_<configProperty>
          <name>APPL_TOP</name>
          <value>/machine/user2/Test/</value>
          <itemMetadata>
            <dataType>STRING</dataType>
            <scope>READ_WRITE</scope>
          </itemMetadata>
        </configProperty>
        |_configProperty-2
        |_configProperty-3
        |_configProperty-4
```

**65.5.3.1 How to Migrate Spawned Job Environment Properties to the Production Environment**

To migrate spawned job environment properties, do the following:

1. Make sure the file cloningclient.jar is located at `<MW_HOME>/oracle_common/jlib`.

2. Make sure the Oracle Enterprise Scheduler plug-in JAR and the registration XML file are located under `<MW_HOME>/Oracle_atgpf/clone/provision`.

3. Pass the location of the original testing environment environment.properties file to the migration plugin by entering the following at the prompt:

   For a Linux operating system, use this command:
   ```bash
   setenv T2P_JAVA_OPTIONS "-Dess.config.dir=\<LocationOfPropsFile>"
   ```

   For a Windows operating system, use these commands:
   ```cmd
   set "T2P_JAVA_OPTIONS=-Dess.config.dir=\<LocationOfPropsFile>"
   cd <MW_HOME>\oracle_common\bin
   ```

4. At the command line, execute the copyConfig script after entering the names of all the servers in the domain or cluster, as shown.

   For a Linux operating system, use this command:
About Configuring a Spawned Job Environment

5. At the command line, execute the `extractPlans` script as shown in the following example.

For a Linux operating system, use this command:

```bash
extractPlans.sh -javaHome /usr/local/packages/jdk16 -archiveLocation /machine/user/11gWLS/DIST/a.jar -planDirLocation /machine/user/11gWLS/EXTRACT -logDirLoc /tmp
```

For a Windows operating system, use this command:

```cmd
extractMovePlan.cmd -javaHome "C:\Program Files\Java\jdk6" -archiveLocation C:\user\11gWLS\DIST\a.jar -planDirLocation C:\user\11gWLS\EXTRACT -logDirLoc C:\tmp
```

This script extracts the `MovePlan.xml` file from the archive to the `planDirLocation`. The extracted XML document contains `environment.properties` entries within the `movableComponent` of type `ESS-EXT`.

6. Change the `MovePlan.xml` file under `planDirLocation` as required by the production environment.

7. At the command line, execute the `pasteConfig` script to recreate the target environment. This step configures the data sources, LDAP, and so on, and starts the servers. Following is an example.

For a Linux operating system, use this command:

```bash
pasteConfig.sh -javaHome /usr/local/packages/jdk16/ -archiveLocation /machine/user/11gWLS/DIST/a.jar -targetDomainLoc /machine/user/11gWLS/user_projects/domains/domain9 -targetMiddlewareHomeLoc /machine/user/11gWLS -movePlanLocation /machine/user/11gWLS/EXTRACT/moveplan.xml -logDirLoc /tmp -silent true
```

For a Windows operating system, use this command:

```cmd
pasteConfig.cmd -javaHome "C:\Program Files\Java\jdk6" -archiveLocation C:\user\11gWLS\DIST\a.jar -targetDomainLoc C:\user\11gWLS\user_projects\domains\domain9 -targetMiddlewareHomeLoc C:\user\11gWLS -movePlanLocation C:\user\11gWLS\EXTRACT\moveplan.xml -logDirLoc C:\\tmp -silent true
```
65.5.4 What Happens When You Configure a Spawned Job Environment

A configured Oracle wallet enables spawned jobs to connect to the database at the command line.

65.6 About Implementing a PL/SQL Scheduled Job

Implementing a PL/SQL scheduled job requires creating a job definition and creating a PL/SQL package.

Related Links

The following documents provide additional information related to subjects discussed in this section:

For more information about implementing a PL/SQL stored procedure scheduled job see the chapter "Creating and Using PL/SQL Jobs" in the Developing Applications for Oracle Enterprise Scheduler.

65.6.1 Standards and Guidelines for Implementing a PL/SQL Scheduled Job

Run subrequests through Oracle Enterprise Scheduler using the Oracle Enterprise Scheduler APIs to access Oracle Enterprise Scheduler.

A PL/SQL stored procedure scheduler job should have a signature with the first two arguments being `errbuf` and `retcode`. The remaining arguments are used as required for defining job parameters. All arguments have a data type of `varchar2`.

65.6.2 About Defining Metadata for a PL/SQL Scheduled Job

Create a job definition as described in About Creating a Job Definition.

PL/SQL jobs require setting an additional property `numberOfArgs` in the job definition. This property identifies the number of job submission arguments (not including the required arguments `errbuf` and `retcode`).

65.6.3 How to Implement a PL/SQL Scheduled Job

Oracle Enterprise Scheduler provides runtime PL/SQL APIs for implementing PL/SQL jobs and running the jobs using Oracle Enterprise Scheduler. A view object is defined and associated with the job definition for the job.

When creating a PL/SQL job, use the `fusion` database user. For information about granting access privileges to database users in the context of Oracle Fusion Applications, see Implementing Oracle Fusion Data Security.

To implement a PL/SQL scheduled job, do the following:

1. Create a PL/SQL package.
2. Deploy the package to a database.
3. Test the package.

65.6.4 What Happens When You Implement a PL/SQL Job

The sample PL/SQL job shown in Example 65-13 provides a signature of a PL/SQL procedure run as a job. The first two arguments to the PL/SQL procedure, `errbuf`
and retcode, are required. The remaining arguments are properties filled in by end
users and passed to Oracle Enterprise Scheduler when the job is submitted.

The example shown in Example 65-13 illustrates a sample PL/SQL job that uses the
PL/SQL API.

The sample shown in Example 65-14 illustrates a PL/SQL job with a subrequest
submission. The no_requests argument identifies the number of subrequests that
must be submitted.

Example 65-13  Running a Job Using the PL/SQL API

```
procedure fusion_plsql_sample(
  -- The first two arguments are required: errbuf and retcode
  -- The errbuf is logged when a job request ends in a warning or error state to
  -- provide a quick indication as to why the job request ended in an error or
  -- warning state.
  -- Job submission arguments, as collected from the view object associated with the
  -- job as configured in the job definition. The view object is used to present a
  -- user interface to end users, allowing them to enter the properties listed in
  -- the following lines of code.
  -- interface. These values are submitted by the end user.
  --
  errbuf    out NOCOPY varchar2,
  retcode   out NOCOPY varchar2,
  run_mode  in  varchar2 default 'BASIC',
  duration  in  varchar2 default '0',
  p_num     in  varchar2 default NULL,
  p_date    in  varchar2 default NULL,
  p_varchar in  varchar2 default NULL) is
begin
  -- Write log file content using FND_FILE API
  FND_FILE.PUT_LINE(FND_FILE.LOG, "About to run the sample program");

  -- Implement the business logic of the job here.
  --
  FND_FILE.PUT_LINE(FND_FILE.OUT, "RUN MODE : " || run_mode);
  FND_FILE.PUT_LINE(FND_FILE.OUT, "DURATION: " || duration);
  FND_FILE.PUT_LINE(FND_FILE.OUT, "P_NUM: " || p_num);
  FND_FILE.PUT_LINE(FND_FILE.OUT, "P_DATE: " || p_date);
  FND_FILE.PUT_LINE(FND_FILE.OUT, "P_VARCHAR: " p_varchar);

  -- Retrieve the job completion status which is returned to Oracle
  -- Enterprise Scheduler.
  errbuf := fnd_message.get("FND", "COMPLETED NORMAL");
  retcode := 0;
end;
```

Example 65-14  Submitting a Subrequest Using the PL/SQL Runtime API

```
procedure fusion_plsql_subreq_sample(
  --
  errbuf    out NOCOPY varchar2,
  retcode   out NOCOPY varchar2,
  no_requests  in  varchar2 default '5',
) is

  req_cnt number := 0;
  sub_reqid number;
  submitted_requests varchar2(100);
  request_prop_table_t jobProp;
```
begin
   -- Write log file content using FND_FILE API
   FND_FILE.PUT_LINE(FND_FILE.LOG, "About to run the sample program with subrequest functionality");

   -- Requesting the PAUSED_STATE property set by job identifies request as having started for the first time or restarting after being paused.
   if ( ess_runtime.get_reqprop_varchar(fnd_job.job_request_id, 'PAUSED_STATE') is null )  -- first time start
      then
         -- Implement the business logic of the job here.
         FND_FILE.PUT_LINE(FND_FILE.OUT, " About to submit subrequests : " ||
         no_requests);

         -- Loop through all the subrequests.
         for req_cnt 1..no_requests loop
            -- Retrieve the request handle and submit the subrequest.
            sub_reqid := ess_runtime.submit_subrequest(request_handle =>
                  fnd_job.request_handle,
                  definition_name => 'sampleJob',
                  definition_package => 'samplePkg',
                  props => jobProp);
            submitted_requests := sub_reqid || ',';
         end loop;

         -- Pause the parent request.
         ess_runtime.update_reqprop_varchar(fnd_job.request_id, 'STATE',
                   ess_job.PAUSED_STATE);

         -- Update the parent request with the state of the subrequest, enabling the job to retrieve the status during restart.
         ess_runtime.update_reqprop_int(fnd_job.request_id, 'PAUSED_STATE',
                   submitted_requests);
      else
         -- Restart the request, retrieve job completion status and return the status to Oracle Enterprise Scheduler.
         errbuf := fnd_message.get("FND", "COMPLETED NORMAL");
         retcode := 0;
      end if;
end;

65.6.5 What Happens at Runtime: How a PL/SQL Job is Implemented

Oracle Enterprise Scheduler calls routines to initialize the context of the PL/SQL job, including PL/SQL global values, local values (such as language and territory), and request-specific values such as request ID and request handle.

The view object associated with the job definition displays a user interface so that end users may fill in values for each property. The Oracle Fusion web application calls Oracle Enterprise Scheduler using the provided APIs and submits the job request. Oracle Enterprise Scheduler runs the job, which calls the context routines and then runs the job logic. The job ends with a retcode value of 0, 1, 2 or 3, representing SUCCESS, WARNING, FAILURE or BUSINESS ERROR, respectively. The Oracle Fusion web application can retrieve the result from Oracle Enterprise Scheduler and display it in the user interface.
65.7 About Implementing a SQL*Plus Scheduled Job

Implementing a SQL*Plus scheduled job involves writing a SQL*Plus script and configuring an environment file for the job.

65.7.1 Standards and Guidelines for Implementing a SQL*Plus Scheduled Job

Run subrequests through Oracle Enterprise Scheduler using the Oracle Enterprise Scheduler APIs to access Oracle Enterprise Scheduler.

65.7.2 How to Implement a SQL*Plus Job

Implementing a SQL*Plus stored procedures job involves writing the SQL*Plus script, storing the script and configuring a spawned job environment.

To implement a SQL*Plus job, do the following:

1. Write the SQL*Plus job as a SQL*Plus script. Include the FND JOB.set_sqlplus_status call so as to report the final job status.

   Include the following in the SQL*Plus scheduled job:

   - FND JOB.set_sqlplus_status: Call to report the final job status. Statuses include:
     - FND JOB.SUCCESS_V: Success.
     - FND JOB.WARNING_V: Warning.
     - FND JOB.FAILURE_V: Failure.
   - FND_FILE routines: Can be used for producing log data and output files.
   - FND_JOB API for request values: API calls are initialized for SQL*Plus jobs.

   Note:
   SQL*Plus jobs must **not** exit.

2. Store the script under PRODUCT_TOP/$APPLSQL.

3. Configure the spawned job environment as described in About Configuring a Spawned Job Environment. Configure the ATGPF_TOP value in the environment.properties file for spawned jobs.

4. Run and test the job.

65.7.3 About Using the SQL*Plus Runtime API

Oracle Enterprise Scheduler provides runtime SQL*Plus APIs for implementing SQL*Plus jobs and running the jobs using Oracle Enterprise Scheduler.

This sample SQL*Plus job provides a signature of a SQL*Plus procedure run as a job. Any necessary arguments are properties filled in by end users and passed to Oracle Enterprise Scheduler when the job is submitted. A view object is defined and
associated with the job definition for the job. The view object is then used to display a user interface so that end users may fill in values for each property. Finally, the sample prints to an output file.

### 65.7.4 What Happens When You Implement a SQL*Plus Job

Example 65-15 shows a sample SQL*Plus scheduled job, which is executed by a wrapper script.

**Example 65-15  Implementing a SQL*Plus Scheduled Job**

```sql
SET VERIFY OFF
SET linesize 132
WHENEVER SQLERROR EXIT FAILURE ROLLBACK;
WHENEVER OSERROR EXIT FAILURE ROLLBACK;
REM dbdrv: none
/* ----------------------------------------------------------------------*/
DECLARE
  errbuf varchar2(240) := NULL;
  retval boolean;
  run_mode varchar2(200) := '&1';
BEGIN
  DBMS_OUTPUT.PUT_LINE(run_mode);
  update dual set dummy = 'Q';
  FND_FILE.PUT_LINE(FND_FILE.LOG, 'Parameter 1 = ' || nvl(run_mode,'NULL'));
  /* print out test message to log file and output file */
  /* by making direct call to FND_FILE.PUT_LINE */
  /* from sql script. */
  FND_FILE.PUT_LINE(FND_FILE.LOG, '-----------------------------------------
  Printing a message to the LOG FILE
  -----------------------------------------
  SUCCESS!
  -----------------------------------------
 ,'
  FND_FILE.PUT_LINE(FND_FILE.OUTPUT,'-----------------------------------------
  Printing a message to the OUTPUT FILE
  -----------------------------------------
  SUCCESS!
  -----------------------------------------
  ',
  retval := FND_JOB.SET_SQLPLUS_STATUS(FND_JOB.SUCCESS_V);
```
65.7.5 What Happens at Runtime: How a SQL*Plus Job Is Implemented

Oracle Enterprise Scheduler calls routines in a wrapper script to initialize the context of the SQL*Plus job, including global values, local values (such as language and territory), and request-specific values such as request ID and request handle. The wrapper script introduces the prologue of commands shown in Example 65-16.

The Oracle Fusion application calls Oracle Enterprise Scheduler using the provided APIs. Oracle Enterprise Scheduler runs the job, and the final job status—SUCCESS, WARNING, BUSINESS ERROR, or FAILURE—is communicated to Oracle Enterprise Scheduler. The Oracle Fusion web application can retrieve the result from Oracle Enterprise Scheduler and display it in the user interface.

Example 65-16  SQL*Plus wrapper script

```
SET TERM OFF
SET PAUSE OFF
SET HEADING OFF
SET FEEDBACK OFF
SET VERIFY OFF
SET ECHO OFF
SET ESCAPE ON
WHENEVER SQLERROR EXIT FAILURE
```

65.8 About Implementing a SQL*Loader Scheduled Job

Implementing a SQL*Loader scheduled job involves creating a SQL*Loader control file and configuring a spawned job environment.

65.8.1 How to Implement a SQL*Loader Scheduled Job

Like all executable jobs (C, SQLPlus, host, and Perl scripts), SQL*Loader jobs require an executable file that is located in the read-only APPLTOP folder. For SQL*Loader jobs, this is the control file. The control file determines which database tables are to be affected by the SQL*Loader command.

It is possible to use a dynamic control file for SQL*Loader subrequest jobs. A SQL*Loader job submitted as a subrequest using a dynamic control file must access the control file from the working directory of the parent job request, rather than the APPLTOP folder.

Keep in mind that the control file and data file must conform to the following SQL*Loader standards:

- Place control files in the $APPLBIN directory under the product TOP. (Subrequests using dynamic control files must instead access the working directory of the parent job request.)
- The control file's name must be the same as the executableName parameter in the job definition.
- Ensure that the full path of the data file's location is the first submit argument to the job.
• Add SQL*Loader options such as `direct=yes`, if needed, as the `sqlldr.directoption` parameter in the job definition.

• Set the job log file as the SQL*Loader LOG parameter so it will automatically contain all SQL*Loader log messages.

• Set the job output file as the SQL*Loader BAD parameter so it will automatically receive any output directed there. Alternatively, you can create two output files for a SQL*Loader job request.

  - `<requestid>_bad.txt`: This is the output of the bad parameter.
  - `<requestid>_discard.txt`: This is the output of the discard parameter, however a discard file is not always generated.

To implement a SQL*Loader scheduled job, do the following:

1. Create a SQL*Loader control file (.ctl).
2. In the parent job request, make sure you have set the system property `workDirectoryRoot` to the working directory of the parent job request.
3. Alternatively, in the case of SQL*Loader subrequests only, configure the job to use a dynamically created control file.
   a. In the job definition for the subrequest, configure the property `execPRWD`. Set this property to `Y` to enable the dynamic control file.
   b. In the parent job request, configure the name of the control file using the `executableName` system property.
4. Enter the full path of the data file as the first submit argument to the job.
5. Store the control file under `PRODUCT_TOP/$APPLBIN`. Skip this step if you are implementing a SQL*Loader subrequest.
6. Configure the spawned job environment as described in About Configuring a Spawned Job Environment.
7. Test the file.

65.8.2 What Happens When You Implement a SQL*Loader Scheduled Job

A sample SQL*Loader scheduled job is shown in Example 65-17.

What Happens When You Implement a SQL*Loader Scheduled Job Subrequest

When the SQL*Loader subrequest completes, the parent request can discover the status of the completed SQL*Loader job request as with any other subrequest. The log and output files are written to the content repository.

What Happens When You Implement a SQL*Loader Scheduled Job Subrequest Using a Dynamic Control File (execPRWD)

When the SQL*Loader subrequest runs, the command line creates the path to the control file by looking for the file named in the system property `executableName` under the directory indicated in the parent job request as the system property `workDirectoryRoot`. 
**Example 65-17  Sample SQL*Loader scheduled job**

This sample control file will upload data from the data file into the `fnd_applcp_test` table, into the columns listed here (`id1`, `id2`, `idn`, `mesg`). See the SQL*Loader documentation for more information about writing control files.

```sql
OPTIONS (silent=(header, feedback, discards))
LOAD DATA
INFILE *
INTO TABLE fnd_applcp_test
APPEND
FIELDS TERMINATED BY ','
(id1,
 id2,
 id3,
 func CHAR(30),
 time SYSDATE,
 action CHAR(30),
 mesg CHAR(240))
```

### 65.9 About Implementing a Perl Scheduled Job

Implementing a Perl scheduled job involves creating a job definition, enabling the Perl job to connect to a database and configuring a spawned job environment.

**Related Links**

The following documents provide additional information related to subjects discussed in this section:

For more information about creating a Perl scheduled job see the chapter "Creating and Using Process Jobs" in *Developing Applications for Oracle Enterprise Scheduler*.

### 65.9.1 How to Implement a Perl Scheduled Job

To implement a Perl scheduled job, do the following:

1. Place the Perl job under the directory `PRODUCT_TOP/$APPLBIN`.

2. Create a job definition for the Perl job, setting the `executableName` parameter to the name of the Perl script. The following functions can be used in the Perl script:
   - `writeln()`: Write a message to the log file.
   - `timestamp()`: Write a timestamped message.

3. To enable the Perl job to connect to a database, use `/@$TWO_TASK` as a connection string without specifying a user name or password.

4. Configure the spawned job environment as described in About Configuring a Spawned Job Environment. The context provides values for the following:
   - `reqid`: The request ID.
   - `outfile`: The full path to the output file.
   - `logfile`: The full path to the log file.
   - `username`: The name of the user submitting the job request.
   - `log`: The log object.
About Implementing a Perl Scheduled Job

5. Implement an exit code for the job, with values of 0, 2 or 3 representing the

following states: success, warning, and business error. All other values represent an
errored state.

6. Test the job.

65.9.2 What Happens When You Implement a Perl Scheduled Job
Example 65-18 shows a sample scheduled Perl job which does the following:
1.

Checks for basic or full mode.

2.

Prints arguments.

3.

Gets the context object of the scheduled job request.

4.

Retrieves contextual information about the scheduled job request, which is stored
in the context object.

5.

Writes the request to the log file.

6.

Prints information as required.

Example 65-18

Perl Scheduled Job

# dbdrv: none
use strict;
(my $VERSION) = q$Revision: 120.1 $ =~ /(\d+(\.\d+)*)/;
print_header("Begin Perl testing script (version $VERSION)");
# check first argument for BASIC or FULL mode
# if not FULL mode, exit successfully without doing anything
if (! $ARGV[0] || uc($ARGV[0]) ne "FULL") {
exit(0);
}
# -- If argument #1 was passed, use it as a sleep time
if ($ARGV[1]) {
if ($ARGV[1] =~ /\D/) {
print "** Argument #1 is not a valid number, unable to sleep!\n\n";
} else {
printf("Sleeping for %d seconds...\n", $ARGV[1]);
sleep($ARGV[1]);
}
}
# -- Arguments
print_header("Arguments");
my $i = 1;
foreach (@ARGV) {
print "Argument #", $i++, ": $_\n";
}
# -- Get the request context object
my $context = get_context();
# -- Use this object to retrieve context information about this request

65-26 Oracle Fusion Applications Developer's Guide


print_header("Context Information");
printf "Request id \t= %d\n", $context->reqid();
printf "User name \t= %d\n", $context->username();
printf "Logfile \t= %s\n", $context->logfile();
printf "Outfile \t= %s\n", $context->outfile();

# -- Writing to the request log file
print_header("Writing to log file");

# -- retrieve a Logfile object from the context
my $log = $context->log();
$log->writeln("This message should appear in the request logfile");
$log->timestamp("This is a timestamped message to the request logfile");

print "Wrote two messages to the request logfile\n";

# -- Print out some useful information

print_header("Environment");
foreach (sort keys %ENV) {
    print "$_=\$ENV{$_}\n";
}

print_header("Perl Information");
print "PROCESS ID = \$\n";
print "REAL USER ID = \$<\n";
print "EFF USER ID = \$>\n";
print "SCRIPT NAME = \$0\n";
print "PERL VERSION = \$\n";
print "OS NAME = \$^O\n";
print "EXE NAME = \$^X\n";
print "WARNINGS ON = \$^W\n";

print "\n\@INC path:\n";
foreach  (@INC) {
    print "\$_\n";
}

print "\nAll loaded perl modules:\n";
foreach (sort keys \%INC) {
    print "\$_ => \$INC{\$_}\n";
}

# -- Exiting the script
# -- The exit status of the script will be used as the request exit status.
# -- A zero exit status is reported as state of success.
# -- An exit status of 2 is reported as a warning state.
# -- An exit status of 3 is reported as a business error state.
# -- Any other exit status is reported as an error state.

print_header("Exiting script with status 0. (Normal completion)" );
exit(0);

sub print_header {
    my $msg = shift;
    print "\n\n", "-" x 40, "\n", $msg, "\n", "-" x 40, "\n";
}
65.10 About Implementing a C Scheduled Job

The main steps required to implement a C scheduled job are as follows:

- Creating a job definition
- Configuring a spawned job environment
- Implementing and testing a C scheduled job

65.10.1 About Defining Metadata for a C Scheduled Job

Create a job definition as described in About Creating a Job Definition.

65.10.2 How to Implement a C Scheduled Job

To implement a C scheduled job, do the following:

1. In a separate function or file rather than in main, implement your required business logic.
   
   Include the following header files:
   
   - afcp.h: This is the header file for Oracle Enterprise Scheduler.
   - afstd.h and afstr.h: These are Oracle Fusion application header files.

2. Call afpend in the business logic function.

3. In the main function, call afprcp, passing to it a pointer to the business logic function.

   The business logic function is called by afprcp, taking the arguments argc, argv, and reqinfo.

4. Save the executable job file to the $APPLICATIONS_BASE/$APPLBIN directory.

5. Configure the spawned job environment, as described in About Configuring a Spawned Job Environment.

   Set both the TOP and APPLBIN variables for your application in the environment.properties file.

65.10.3 About the Scheduled C Job API

Several C functions are available for use in developing Oracle Fusion applications, while several others are not. Table 65-2 and Table 65-3 list the available and unavailable functions.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>afprcp</strong></td>
<td>Run C program. The recommended API for writing a C program. The main OC file should call this function to run the program logic. It initializes the context and calls the program.</td>
</tr>
<tr>
<td></td>
<td>int afprcp (uword argc, text **argv, afsqlopt *options, afpfcn *function);</td>
</tr>
<tr>
<td><strong>afpend</strong></td>
<td>End C program. All programs must call this to signal the completion of the program. The program should pass completion status and message if necessary.</td>
</tr>
<tr>
<td></td>
<td>Indicate completion status with the following constants:</td>
</tr>
<tr>
<td></td>
<td>• FDP_SUCCESS: Success</td>
</tr>
<tr>
<td></td>
<td>• FDP_WARNING: Warning</td>
</tr>
<tr>
<td></td>
<td>• FDP_ERROR: System Error</td>
</tr>
<tr>
<td></td>
<td>• FDP_BIZERR: Business Error</td>
</tr>
<tr>
<td></td>
<td>boolean afpend (text *outcome, dvoid *handle, text *compmesg);</td>
</tr>
<tr>
<td><strong>fdpfrs</strong></td>
<td>Find request status. For a given request, retrieve the status. The following are possible request states:</td>
</tr>
<tr>
<td></td>
<td>• ESS_WAIT_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_READY_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_RUNNING_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_COMPLETED_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_BLOCKED_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_HOLD_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_CANCELLING_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_EXPIRED_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_CANCELLED_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_ERROR_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_WARNING_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_SUCCEEDED_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_PAUSED_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_PENDING_VALID_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_VALID_FAILED_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_SCHEDULE_ENDED_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_FINISHED_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_ERROR_AUTO_RETRY_STATE</td>
</tr>
<tr>
<td></td>
<td>• ESS_MANUAL_RECOVERY_STATE</td>
</tr>
<tr>
<td></td>
<td>afreqstate fdpfrs (text *request_id, text *errbuf);</td>
</tr>
</tbody>
</table>
**Table 65-2  (Cont.) C Functions Available for Developing Oracle Fusion Applications**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
</table>
| fdpgret  | Get the error type of a specific job request ID. The following are possible error types:  
  - ESS_UNDEFINED_ERROR_TYPE  
  - ESS_SYSTEM_ERROR_TYPE  
  - ESS_BUSINESS_ERROR_TYPE  
  - ESS_TIMEOUT_ERROR_TYPE  
  - ESS_MIXED_NON_BUSINESS_ERROR_TYPE  
  - ESS_MIXED_BUSINESS_ERROR_TYPE  
  afreqstate fdpgret (text *request_id, text *status, text *errbuf); |
| fdpgrs   | Get request status. For a given request, retrieve the current status and completion text.  
  afreqstate fdpgrs (text *request_id, text *status, text *errbuf); |
| fdplck   | Lock table. Locks the desired table with the specified lock mode and NOWAIT. |
| fdpscp   | Legacy API for concurrent programs. All new concurrent programs should use afrmcp.  
  boolean fdpscp (sword *argc, text **argv[], text args_type, text *errbuf); |
| fdpwrt   | Routines for creating log/output files and writing to files. These are routines concurrent programs should use for writing to all log and output files. |

**Table 65-3  C Functions Not Available for Developing Oracle Fusion Applications**

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>fdpgoi</td>
</tr>
<tr>
<td>fdpgpn</td>
</tr>
<tr>
<td>fdpgrc</td>
</tr>
<tr>
<td>fdpimp</td>
</tr>
<tr>
<td>fdpldr</td>
</tr>
<tr>
<td>fdpperl</td>
</tr>
<tr>
<td>fdprep</td>
</tr>
<tr>
<td>fdprpt</td>
</tr>
<tr>
<td>fdprsg</td>
</tr>
<tr>
<td>fdpscr</td>
</tr>
<tr>
<td>fdpsql</td>
</tr>
<tr>
<td>fdpstp</td>
</tr>
</tbody>
</table>
65.10.4 About Testing a C Scheduled Job

When developing a C job, it is possible to test the job by running it from a command line interface.

Running a C job from the command line involves the following main steps:

- Invoking the job
- Obtaining a database connection and setting the runtime context by passing special arguments.
- Passing any program-specific parameters at the command line.
- To run a C job from the command line, use the syntax shown in Example 65-19 to run a C job from the command line for testing purposes.

where

<heavyweight user connection string> is the username/password@TWO_TASK pair used to connect to the database

<lightweight user name> is the name of the lightweight user submitting the job. This value is used to set the user context in the database connection.

<flag> must be set to 'L' for lightweight user.

An example illustrating running a C job from the command line is shown in Example 65-20.

**Example 65-19  Syntax for Running a C Job from the Command Line**

```bash
%program <heavyweight user connection string> <lightweight username> <flag> <job parameters> ...
```

**Example 65-20  Running a C Job from the Command Line for Testing Purposes**

```bash
program username/password@my_db MYUSER L <parameter1> <parameter2> ....
```

65.10.5 What Happens When You Implement a C Scheduled Job

The sample C job shown in Example 65-21 uses afprcp to initialize and obtain a database connection. It uses both Pro*C and afupi.

**Example 65-21  Using the C Runtime API**

```c
#ifndef AFSTD
#include <afstd.h>
#endif

#ifndef AFSTR
#include <afstr.h>
#endif

#ifndef AFCP
#include <afcp.h>
#endif

#ifndef SQLCA
#include <sqlca.h>
#endif
```
About Implementing a C Scheduled Job

#ifndef AFUPI
#include <afupi.h>
#endif
#ifndef FDS
#include <fds.h>
#endif
boolean testupi()
{
text *sqltext;
text buffer[ERRLEN];
text os_user[31];
text session_user[31];
text db_name[31];
aucursor *use_curs;
word
errcode;
os_user[0] = session_user[0] = db_name[0] = (text)'\0';
sqltext = (text*) "SELECT sys_context('USERENV','DB_NAME',30), sys_context('US
ERENV','SESSION_USER',30), sys_context('USERENV','OS_USER',30) from dual";
use_curs = NULLCURSOR;
use_curs = afuopen (NULLHOST, NULLCURSOR, (dvoid *)
sqltext,
UPISTRING);
if (use_curs == NULLCURSOR) {goto upierror;}
afudefine(use_curs, 1, AFUSTRING, (dvoid *)db_name, 31);
afudefine(use_curs, 2, AFUSTRING, (dvoid *)session_user, 31);
afudefine(use_curs, 3, AFUSTRING, (dvoid *)os_user, 31);
if (!afuexec (use_curs, (uword)1, (uword)1, CSTATHOLD|CSTATEXACT) ||
(errcode = afuerror (NULLHOST, (text *) NULL, 0)) != ORA_NORMAL) {
goto upierror;
}
DISCARD afurelease (use_curs);
DISCARD sprintf((char *)buffer, "%s as %s@%s", os_user,
session_user, db_name);
DISCARD fdpwrt(AFWRT_OUT | AFWRT_NEWLINE, buffer);

return TRUE;
upierror:
if (use_curs != NULLCURSOR)
DISCARD afurelease (use_curs);
DISCARD fdpwrt(AFWRT_LOG | AFWRT_NEWLINE, "Error in testupi");
return FALSE;
}
void testrpc()
{
text buffer[256];

65-32 Oracle Fusion Applications Developer's Guide


EXEC SQL BEGIN DECLARE SECTION;

VARCHAR os_user[31];
VARCHAR session_user[31];
VARCHAR db_name[31];

EXEC SQL END DECLARE SECTION;

buffer[0] = os_user.arr[0] = session_user.arr[0] = db_name.arr[0] = '\0';

EXEC SQL SELECT sys_context('USERENV','DB_NAME',30),
 sys_context('USERENV','SESSION_USER',30),
 sys_context('USERENV','OS_USER',30)
 INTO :db_name, :session_user, :os_user
 from dual;

nullterm(os_user);
nullterm(session_user);
nullterm(db_name);

DISCARD sprintf((char *)buffer, "%s as %s@%s", os_user.arr,
 session_user.arr, db_name.arr);

DISCARD fdpwrt(AFWRT_OUT | AFWRT_NEWLINE, buffer);
}

sword cptest(argc, argv, reqinfo)
/* ARGSUSED */
sword argc;
text *argv[];
dvoid *reqinfo;
{
  ub2 i;
text errbuf[ERRLEN+1];

  /* Write to the log file */
  DISCARD fdpwrt(AFWRT_LOG | AFWRT_NEWLINE, (text *)"Test Success");
  /* Write to the out file */
  DISCARD fdpwrt(AFWRT_OUT | AFWRT_NEWLINE, (text *)"Test Args:");
  /* Loop through argv and write to the out file. */
  for ( i=0; i<argc; i++)
    DISCARD fdpwrt(AFWRT_OUT | AFWRT_NEWLINE, argv[i]);
  /* Call the Oracle Fusion Applications function afpoget to return the value */
  /* of a profile option called SITENAME and write the results to the error */
  /* buffer. */
  DISCARD afpoget((text *)"SITENAME", errbuf);
  /*Write the value to the output file. */
  DISCARD fdpwrt(AFWRT_OUT | AFWRT_NEWLINE, errbuf);
  /* Connect to the database and run a SELECT against the database. Creates a */
  /* string and writes the returned data to the output file. Uses prc APIs. */
  testrpc();
  /* Open a cursor for the SELECT statement, defines variables to collect data */
  /* upon running statement, and executes SELECT. Creates a string which it */
  /* writes to the output file. Uses afupi APIs. */
  testupi();
  /* Writes the string "Test Completed." to the output file. */
  DISCARD fdpwrt(AFWRT_OUT | AFWRT_NEWLINE, (text *)"Test Completed.");
  /* Call afpend to identify the exit status, which in this case is successful. */
  /* Other possible values are FDP_WARNING, FDP_ERROR and FDP_BIZERR. The */
  /* reqinfo originally passed to cptest is passed here. Optionally, additional */
  /* text can be passed here, for example explaining the outcome of the exit */
}
65.10.6 What Happens at Runtime: How a C Scheduled Job Is Implemented

When Oracle Enterprise Scheduler runs a C job, afprcp() runs first to initialize the context and obtain the database connection. The function afprcp() then calls the function containing the program logic. Oracle Enterprise Scheduler runs the job, and the result of the job is returned to Oracle Enterprise Scheduler. The Oracle Fusion application can retrieve the result from Oracle Enterprise Scheduler and display it in the user interface.

Note:

Wallet configuration is required for the client ORACLE_HOME to obtain the database connection. The operating system environment in which the job runs (including the location of the client ORACLE_HOME, which is also required) is set in the environment.properties file. The environment.properties file must be configured and placed in the config/fmwconfig directory under the domain.

You can add your own environment variables by creating an env.custom.properties file in the same directory. Variables you define in this file take precedence over those in the environment.properties file.

Similarly, you can set server-specific environment variables with environment.properties and env.custom.properties files in the server config directory.

65.11 About Implementing a Host Script Scheduled Job

Arguments submitted for a host script job request are passed to the script at the command line. Host scripts may access the standard environment variables to get REQUESTID, LOG_WORK_DIRECTORY, OUTPUT_WORK_DIRECTORY, and so on. Script output is redirected to the request log file by default.

Use the following steps when implementing a host script job:

- Complete the steps for configuring a spawned job as described in About Configuring a Spawned Job Environment.
- Create one script file each for Unix and Windows platforms. Name each script file the same as executableName parameter in the job definition. For example, if
your executableName is "myscript", the script files would be called
myscript.sh (on Unix platforms) and myscript.cmd (on Windows).

• Put host scripts in the $APPLBIN directory under the product TOP.
• The script should exit with one of the following exit codes (anything else is
considered a SYSTEM ERROR):
  – 0 for SUCCESS
  – 2 for WARNING
  – 3 for BUSINESS ERROR

65.12 About Implementing a Java Scheduled Job

For more information about implementing Java scheduled jobs, see the chapter "Using
Oracle JDeveloper to Generate an Oracle Enterprise Scheduler Application" in
Developing Applications for Oracle Enterprise Scheduler.

65.12.1 About Defining Metadata for a Scheduled Java Job

Create a job definition as described in About Creating a Job Definition.

65.12.2 About Using the Java Runtime API

For information about the Java runtime API, see the Oracle Fusion Applications Java API
Reference for Oracle Enterprise Scheduler Service.

You can access the Oracle Fusion Middleware Extensions for Applications Message
and Profile objects directly, using those APIs which handle the service accessing
themselves.

65.12.3 About Cancelling a Scheduled Java Job

You can cancel a scheduled Java job by implementing the Cancellable interface.

The Cancellable implementation in Example 65-22 checks as logic progresses to see
if the job has been canceled. If it has, the code cleans up after itself before exiting.

Example 65-22  Handling a Job Cancellation Request

import oracle.as.scheduler.Cancellable;
import oracle.as.scheduler.Executable;
import oracle.as.scheduler.ExecutionCancelledException;
import oracle.as.scheduler.ExecutionErrorException;
import oracle.as.scheduler.ExecutionPausedException;
import oracle.as.scheduler.ExecutionWarningException;
import oracle.as.scheduler.RequestExecutionContext;
import oracle.as.scheduler.RequestParameters;

public class MyExecutable
  implements Executable, Cancellable
{
    private volatile boolean m_cancel = false;

    public void execute( RequestExecutionContext reqCtx,
                         RequestParameters reqParams )
      throws ExecutionErrorException, ExecutionWarningException,
             ExecutionPausedException, ExecutionCancelledException
{  
  // Do some work and check if this request has been canceled.  
  // ... work ...  
  checkCancel(reqCtx);  
  
  // Do more work and check if this request has been canceled.  
  // ... work ...  
  checkCancel(reqCtx);  
  // Finish work.  
  // ... work ...  
}

// Set flag that the app logic should check periodically to  
// determine if this request has been canceled.  
public void cancel()  
{  
  m_cancel = true;  
}

// Check if request has been canceled. If not, do nothing.  
// Otherwise, do any cleanup work that may be needed for  
// this request and end by throwing an ExecutionCancelledException.  
private void checkCancel(RequestExecutionContext reqCtx)  
  throws ExecutionCancelledException  
{  
  if (m_cancel)  
  {  
    // Do work any cleanup work that may be needed  
    // before ending this executable.  
    // ... cleanup work ...  
    String msg = "Request " + reqCtx.getRequestId() +  
    " was cancelled.";
    
    throw new ExecutionCancelledException(msg);  
  }  
  
}  

65.12.4 What Happens at Runtime: How a Java Scheduled Job Is Implemented

Oracle Enterprise Scheduler initializes the context of the job. The Oracle Fusion application calls Oracle Enterprise Scheduler using the provided APIs. Oracle Enterprise Scheduler runs the job, and a result of success or failure is returned to Oracle Enterprise Scheduler. The Oracle Fusion application can retrieve the result from Oracle Enterprise Scheduler and display it in the user interface.

65.13 About Elevating Access Privileges for a Scheduled Job

Oracle Enterprise Scheduler executes jobs in the user context of the job submitter at the scheduled time. Some scheduled jobs require access privileges that are different from those of the submitting user. However, information regarding the submitter of the scheduled job must be retrievable for auditing purposes.

In Oracle Enterprise Scheduler, it is prohibited to run a job in the context of a user other than the submitting user with the runAs property. Doing so would be considered a security breach. Using an application identity enables running a job with different access privileges from those allotted to the submitting user.

Application identity is a SOA and Java Platform Security (JPS) concept that addresses the requirement for escalated privileges in completing an action. The application installer creates an application identity in Oracle Identity Management Repository.
For more information, see the following chapters:

- Implementing Application User Sessions
- Implementing Oracle Fusion Data Security
- Implementing Function Security

### 65.13.1 How to Elevate Access Privileges for a Scheduled Job

The Oracle Enterprise Scheduler job system property `SYS_runasApplicationID` enables elevating access privileges for completing a scheduled job.

To elevate access privileges, do the following:

1. Create a job definition, as described in About Creating a Job Definition.
2. Under the Parameters section, add a parameter called `SYS_runasApplicationID`.
3. In the text field for the `SYS_runasApplicationID` property, enter the application ID under which you want to run the job, as shown in Figure 65-1.

The input string must be a valid `ApplicationID` value that exists when the job executes.

**Figure 65-1  Defining the runAs User for the Job**

![Job Definition](image)

You can retrieve the executing user by running either of the methods shown in Example 65-23 and Example 65-24.

**Example 65-23  Retrieving the Executing User with getRunAsUser()**

```java
requestDetail.getRunAsUser()
```
Example 65-24  Retrieving the Executing User with getRequestParameter()

String sysPropUserName =
    (String) runtime.getRequestParameter(h, reqid, SystemProperty.USER_NAME);

Given a request ID, you can retrieve the submitting and executing users of a job request.

65.13.1.1 About Retrieving the Submitting and Executing Users of a Job Request in Oracle Enterprise Scheduler RuntimeService Enterprise JavaBeans Object

Example 65-25 shows a code sample for retrieving the submitting and executing users of a job request using the Oracle Enterprise Scheduler RuntimeService Enterprise JavaBeans object.

Example 65-25  Retrieving the Submitting and Executing Users of a Job Request Using the RuntimeService Enterprise JavaBeans Object

// Lookup runtimeService
RequestDetail requestDetail = runtimeService.getRequestDetail(h, reqid);
String runAsUser = requestDetail.getRunAsUser();
String submitter = requestDetail.getSubmitter();

65.13.1.2 About Retrieving the Submitting and Executing Users of a Job Request from Within an Oracle Fusion Application

Example 65-26 shows a code sample for retrieving the submitting and executing users of a job request from within an Oracle Fusion application.

Example 65-26  Retrieving the Submitting and Executing Users of a Job Request from an Oracle Fusion application

import oracle.apps.fnd.applcore.common.ApplSessionUtil;
// The elevated privilege user name.
ApplSessionUtil.getUserName()
// The submitting user.
ApplSessionUtil.getHistoryOverrideUserName()

65.13.2 How Access Privileges Are Elevated for a Scheduled Job

When a job request schedule executes, Oracle Enterprise Scheduler:

1. Validates the submitter's execution privileges on the job metadata.

2. Retrieves the application identity information from the job metadata. If the job metadata does not specify an application identity for the job, Oracle Enterprise Scheduler executes the job in the context of the job submitter.

   • Java job: An FND session is established as the user with elevated privileges.

       The executing user is taken from the current subject as viewed from the job logic.
Note:
Oracle Enterprise Scheduler does not directly support invoking a web service or composite. If your job logic invokes a web service or composite, you must write the client code logic in your job, establish a connection and propagate the job submitter information as data for auditing purposes. For an asynchronous web service call, the job must wait for a response.

- **Spawned C job:** An application user session is established as the executing user. The submitter information is an attribute of the application user session. The spawned job executes as the operating system user who starts Oracle WebLogic Server.
- **PL/SQL job:** An FND session is established as the executing user. The submitter information is attribute of the FND session. The job runs in the context of the FND session in the RDBMS job scheduler.

3. Executes the job logic.

### 65.13.3 What Happens When Access Privileges Are Elevated for a Scheduled Job
Oracle Enterprise Scheduler validates the user's execution privileges on the job metadata. If so, the user context is captured and stored in the Oracle Enterprise Scheduler database as the submitting user, and the request is placed in the queue.

### 65.14 About Enabling Multiple Language Support for Jobs
Multiple language jobs enable repeatedly running an end-user's submitted request in different languages such that the data processes accordingly.

Enabling multiple language support for jobs involves the following steps:

1. Identify a set of languages in which the job runs.
2. Develop a multiple-language support function (PL/SQL, Java, or ADF Business Components) and register it with concurrent processing.
3. Associate the multiple language support function with one or more programs.

The following multiple language support functions can be defined:

- **About Defining Java Multiple Language Support Functions**
- **About Defining PL/SQL Multiple Language Support Functions**
- **How to Define ADF Business Components Multiple Language Support Functions**

### 65.14.1 About Defining Java Multiple Language Support Functions
Java multiple language support functions can be defined in the client interface.

#### 65.14.1.1 About Defining Concurrent Processing in the Client Interface Using Java
The client interface for the Java multi-language function should be a fully qualified class name that implements the interface shown in Example 65-27.
The `ExecutionLanguageList` interface is available in the `oracle.apps.fnd.applcp.mls.server` package.

This class implements a default constructor. Parameters to the program are passed through the parameters array. Parameter n0 contains the parameter token and parameter n1 contains the parameter value. The returned string array is a valid list of languages, such that both positional and named parameters are supported for clients. The connection object should be valid and secured to the relevant schema.

A sample class is shown in Example 65-28.

**Example 65-27  The ExecutionLanguageList Interface**

```java
public interface ExecutionLanguageList {
    public String[] getLanguageList(Connection conn, String[][] parameters);
}
```

**Example 65-28  Defining Concurrent Processing in the Client Interface Using Java**

```java
public class MyLanguageList implements ExecutionLanguageList {
    public String[] getLanguageList(Connection conn, String[][] parameters) {
        String[] myStrArray;
        ...
        return myStrArray;
    }
}
```

### 65.14.1.2 About Using the Java Support API for Multiple Language Functions

Currently, no Java support APIs are provided.

### 65.14.2 About Defining PL/SQL Multiple Language Support Functions

PL/SQL multiple language support functions can be defined in the client interface or using the PL/SQL support API.

#### 65.14.2.1 About Defining Concurrent Processing in the Client Interface Using PL/SQL

The client interface for a PL/SQL multi-language support function should be a PL/SQL function name optionally prefixed with a package name and separated by a period. The PL/SQL function does not take any input parameters and returns a `VARCHAR2` value. The returned `VARCHAR2` value includes a list of acceptable comma separated languages (such as those listed in the `FND_LANGUAGES` property). If a returned language is not listed in the `FND_LANGUAGES` property or the language is not installed, the caller of the client interface can raise an exception or cause the request to fail either at submission time or at runtime.

A sample return value for this function is shown in Example 65-30:

In the previous example, the concurrent processing program's multi-language support function property is `PROD_MLS_PKG.GET_LANGUAGES`.

**Example 65-29  A Sample PL/SQL Multiple Language Support Function**

```sql
FUNCTION PROD_MLS_PKG.GET_LANGUAGES RETURN VARCHAR2;
```

**Example 65-30  Sample Return Value**

```
US,F,AR,KO
```
65.14.2.2 About Defining Concurrent Processing Using the PL/SQL Support API

The API package FND_REQUEST_INFO is provided to enable interrogating the current context and act accordingly.

The following functions and procedures are provided:

- **GET_PROGRAM**: Returns the name of the program being executed and the application to which it belongs. This is useful in cases where the same multi-language support function is used for multiple programs.

- **GET_PARAMETER**: Given a parameter number from the ordered list of parameters, returns the parameter name for the current request.

**Example 65-31  GET_PROGRAM Syntax**

```sql
procedure GET_PROGRAM (prog_name out varchar2, prog_app_name out varchar2);
```

**Example 65-32  GET_PARAMETER Syntax**

```sql
function GET_PARAMETER (param_num in number) return varchar2;
```

65.14.3 How to Define ADF Business Components Multiple Language Support Functions

A list of supported languages can also be retrieved from an ADF Business Components view object.

To define ADF Business Components multiple language support functions, do the following:

1. Create a view object that returns a list of supported languages, with the first column in the view object listing the language code.

2. Configure the view object to extend the interface oracle.apps.fnd.applcp.mls.server.MLSViewObject. The view object implements the method initQuery(String[][] parameters), which accepts a string array as its parameter. The string array contains the parameter tokens and values submitted to the request in the same sequence in which they are registered. The initQuery() method should then use one or more of these parameters to create or append the where clause of the view object, and execute the corresponding query.

3. Associate the view object with an application module.

4. Create an ADF Business Components multiple language support function and set the value to the fully qualified package name, followed by the Application Module name and the view object name.

A sample value for the multiple language support function is shown in Example 65-33:

```
The view object name should be specific to the view object instance, such that calling the findViewObject("MyLanguageVO1") method returns the view object instance to the caller.
Example 65-33  Value for the ADF Business Components Multiple Language Support Function

oracle.apps.fnd.cp.MySampleAM.MyLanguageVO1.

65.14.3.1 What Happens When You Define an ADF Business Components Multiple Language Support Function

When a program executes this multi-language support function, the following occurs:

1. The Application Module specified in the function is located.
2. The method findViewObject() is called on the module to get a handle for the view object.
3. The method initQuery(String[][]) is called on the view object.

65.14.3.2 About ADF Business Components Support APIs

Currently, no support APIs are provided.

65.14.4 About Processing Output for Multiple Language Jobs

The output generated from each request may need to be formatted into readable output formats. For example, XML output may need to be formatted to be readable to end users. Output can be formatted using templates during postprocessing. Usually, a particular formatting template is associated with a program for a given language. If a program is not associated with a template, end users can specify a formatting template for one or more languages when submitting a job. If no template is specified for a specific language, a designated default template should be applied for that language.

The generated output can be sent to zero or more printers based on the language being run. In some cases, printers may not have the fonts required to print documents in a particular language. Alternatively, all print output can be sent to the same printer. The printing interfaces to concurrent processing handle sending the output to the correct printer based on the parameters specified by the user.

Users who submit requests can also choose to specify completion notifications to be automatically sent to zero or more recipients. This notification may contain a link to the output, for example. In the case of multiple language requests, users can specify zero or more recipients of notifications for all languages or for an individual language. As in the case of printing, the notification interfaces handle the delivery of notifications according to the sender’s instructions.

The list of languages that can be selected by the user is driven by the languages supported by the installed software. Any function written for multiple language support returns a valid subset of languages from this list.

65.14.5 What Happens When You Enable Multiple Language Support for Jobs

The multiple language support function returns a list of supported languages in a format supported by concurrent processing. The programs associated with the multiple language support function execute multiple times for each request, once for each returned language.

If a program has no associated multiple language support function but is aware of the language context, users can explicitly choose one or more languages in which the program runs when submitting a request.
When a job is submitted for a program with an attached multiple language support function, the submission interface evaluates the function to retrieve the list of languages. This language list is used to drive postprocessing options such as publishing and printing.

65.14.6 What Happens at Runtime: How Multiple Language Support Is Enabled for Jobs

Multiple language support functions are not evaluated at runtime. Convenience messages may be provided to users when attempting to submit a program already scheduled for multiple language support execution if there is an overlap in the set of languages in the two submissions.

65.14.7 Standards and Guidelines Regarding Multiple Language Support Functions

The following standards and guidelines apply to using multi-language support functions:

- Runtime evaluation of multi-language support functions is no longer supported.
- Multiple language support functions can be registered in the program definition user interface.
- Multiple language support functions are evaluated at the time of request submission.

65.15 About Creating an Oracle ADF User Interface for Submitting Job Requests

When implemented as part of an Oracle Fusion application, the Oracle ADF user interface enables end users to submit job requests.

Related Links

The following documents provide additional information related to subjects discussed in this section:

- For more information about configuring security certificates, see the chapter "Managing Keystores, Wallets, and Certificates" in the Administering Oracle Fusion Middleware.

- For more information about creating task flows and binding them to an Oracle ADF user interface, see the following chapters in Developing Fusion Web Applications with Oracle Application Development Framework:
  - "Getting Started with ADF Task Flows"
  - "Working with Task Flow Activities"
  - "Using Task Flows as Regions"

- For more information about creating an ADF Business Components view object, see the chapters "Defining SQL Queries Using View Objects" and "Advanced View Object Techniques" in Developing Fusion Web Applications with Oracle Application Development Framework.

- For more information about passing parameters to the Oracle ADF task flow, see the chapter "Using Parameters in Task Flows" in Developing Fusion Web Applications with Oracle Application Development Framework.
• For more information about deploying Oracle Enterprise Scheduler applications, see the section "Deploying and Running the EssDemoApp Application" in the Developing Applications for Oracle Enterprise Scheduler.

65.15.1 How to Create an Oracle ADF User Interface for Submitting Job Requests

The Oracle ADF UI enables end users to submit job requests. End users can enter complex data types for the arguments of descriptive and key flexfields. The Parameters tab in the Oracle ADF UI interface allows end users to enter parameters to be used when submitting the job request.

Flexfields display in a separate task flow region. This region is a child task flow of the parent task flow displayed in the Parameters tab.

---

**Note:**

Define customization layers and authorize runtime customizations to the adf-config.xml file as described in Creating Customizable Applications.

---

To create a user interface for submitting job requests, do the following:

1. Create a new Oracle Fusion web application by clicking New Application in the Application Navigator and selecting Fusion Web Application (ADF) from the Application Templates drop-down list.
   Model and ViewController projects are created within the application.

2. Right-click the Model project and select Project Properties > Libraries and Classpath > Add Library.

3. From the list, select the following libraries, as shown in Figure 65-2.
   • Applications Core
   • Applications Concurrent Processing
   • Enterprise Scheduler Extensions
4. Right-click the View Controller project and select Project Properties > Libraries and Classpath > Add Library.

Add the library Applications Core (ViewController), as shown in Figure 65-3.

5. In the Project Properties dialog, in the left-hand pane, click Business Components.
6. The Initialize Business Components Project window displays. Click the Edit icon to create a database connection for the project. Fill in the database connection details as follows:

- **Connection Exists in:** Application Resources
- **Connection Type:** Oracle (JDBC)
- **User name/Password:** Fill in the relevant user name and password for the database.
- **Driver:** thin
- **Host Name:** Enter the host name of the database server.
- **JDBC port:** Enter the port number of the database.
- **SID:** The unique Oracle system ID for the database.

Click OK.

7. In the file `weblogic.xml`, import the `oracle.applcp.view` library.

8. In the file `weblogic-application.xml`, import the following libraries:
   - `oracle.applcore.attachments` (for ESS-UCM)
   - `oracle.applcp.model`
   - `oracle.applcp.runtime`
   - `oracle.ess`
   - `oracle.sdp.client` (for notification)
   - `oracle.ucm.ridc.app-lib` (for ESS-UCM)
   - `oracle.webcenter.framework` (for ESS-UCM)
   - `oracle.xdo.runtime`
   - `oracle.xdo.service.client`
   - `oracle.xdo.webapp`

   The libraries `oracle.applcp.model` and `oracle.applcp.view` are deployed as part of the installation while running the `config.sh` wizard.

9. Create a new Java Server Pages XML (JSPX) page for the ViewController project by right-clicking ViewController and selecting **New > Web Tier > JSF > JSF JSP Page**.

10. Create a new File System connection. In the Resource Palette, right-click File System, select **New File System Connection**, and do the following:

   a. Provide a connection name and directory path for the Oracle ADF Library files (`<jdev_install>/jdev/oaext/adflib`).

   b. Click **Test Connection** and click **OK** after the connection succeeds.
11. Expand the contents of the SRS-View.jar file to display the list of available task flows that can be used in the application, as shown in *Figure 65-4.*

*Figure 65-4  Displaying the List of Available Task Flows*

12. To include the job request submission page in the application, select the ScheduleRequest-taskflow item from the Resource Palette and drop it onto the Java Server Faces (JSF) page in the area where you want to create a call to the task flow. Create the task flow call as a link or button.

For example, to invoke the job request submission page from within a dialog in the application, do the following:

a. From the Component Palette, drag and drop a Link onto the form in the JSPX page.

b. In the Property Inspector, configure the behavior of the link to the value showpopup.

c. From the Component Palette, drag and drop a Popup component with a dialog component onto the form.

d. To enable submitting a job request, drag and drop the ScheduleRequest-taskflow item onto the dialog component as a dynamic region.

To enable submitting a job set request, drag and drop the ScheduleJobset-taskflow item onto the dialog component.

*Figure 65-5* displays the task flows in the Resource Palette.
13. When prompted, add the required library to the ViewController project by clicking Add Library. Save the JSF page.

14. Edit the task flow binding. Define the following parameters for the task flow, as shown in Figure 65-6.

   a. **jobdefinitionname**: Enter the name of the job definition to be submitted. This is not the name that displays. This is the job definition defined in About Creating a Job Definition. Required.

   b. **jobdefinitionpackagename**: Enter the package name under which the job definition metadata is stored. This should be the namespace path appended to the package name, for example /oracle/ess/Scheduler. The namespace path typically begins with a forward slash (“/”), but should have no forward slash at the end. Required.

   c. **centralui**: When setting this parameter to true, then the task flow UI does not display the header section containing the name, description and basic Oracle BI Publisher actions (such as email, print and notify). This parameter must be a Boolean value. Optional.

   d. **pageTitle**: When passed, the task flow will render this passed String value as the page title. The pageTitle value is currently configured to be truncated at 30 characters. Optional.

   e. **requireRootOutcome**: If true is passed as the value, then the task flow will generate a value of root-outcome when the user clicks the Submit or Cancel buttons. By default, the task flow generates a value of parent-outcome. Optional.

   f. **requestparametersmap**: Enter the name of the map object variable that contains the parameters required for the job request submission. If this parameter is filled in, the Parameters tab in the request scheduling submission page will not prompt end users to enter parameters for executing
the request. The map can be passed to the task flow as a parameter. Typically, this parameter takes the data type `java.util.Map` in which keys are parameter names and values are parameter values. For example, if you will be using a `paramsMap` object in the `pageFlowScope` context, you might enter a `requestparametersmap` value of `#{pageFlowScope.paramsMap}`. Optional.

In the page that holds the task flow region in the job request submission page, set the following property for the popup window that opens the job request submission page window: `contentDelivery = immediate`.

In the page definition file of the page that contains the task flow region, set the following property for the task flow: `Pagedef > executables > taskflow > Refresh=IfNeeded`.

15. If you are using a map to pass parameters to the task flow (as shown in Figure 65-6, the map is called `requestparametersmap`), create a new task flow parameter, such as the `paramsMap` object in the `pageFlowScope` element of a page flow.

![Figure 65-6 Defining Parameters for the Task Flow](image)

These values can be accessed in the job executable, for example from the `RequestParameters` object in the case of a Java job. Example 65-34 illustrates passing the values stored in the `RequestParameters` object to a Java job. This code is used in the class that implements the `oracle.as.scheduler.Executable` interface.

**Note:**

When using a `requestparametersmap` object, set the following properties for the popup window within which the task flow is started.

- Set Content Delivery to `Immediate`.
- In the page definition XML file for the page that contains the region, select `PageDef > Executables > taskflow > set Refresh = IfNeeded`.

16. If the job is defined with properties that must be filled in by end users, the user interface allows end users to fill in these properties before submitting the job...
request. For example, if the job requires a start and end time, end users can fill in the desired start and end times in the space provided by the user interface.

The properties that are filled in by end users are associated with a view object, which in turn is associated with the job definition itself. When the job runs, Oracle Enterprise Scheduler accesses the view object to retrieve the values of the properties.

If using a view object to pass parameters to the job definition, do the following:

a. Create a view object called TestVO using a query such as the one shown in Example 65-35.

b. Specify control UI hints, for example set the display label for Attribute1 to Run Mode and for Attribute2 to Duration.

The parameters tab in the job request submission UI renders with the input fields Run Mode and Duration.

c. To render the Parameters tab in the job request submission UI, add the DynamicComponents 1.0 library as follows. Right-click ViewController and select Project Properties > JSP Tag Libraries > Add. In the Choose Tag Libraries window, select the library DynamicComponents 1.0 and click OK. Figure 65-7 displays the Choose Tag Libraries window.

Figure 65-7 Adding the Library DynamicComponents 1.0

17. In the JSF application you created, create another project called Scheduler. Select File > New, and choose General > Empty Project. This project will be used to create Oracle Enterprise Scheduler metadata and job implementations.

18. In the Scheduler project, add the Oracle Enterprise Scheduler Extensions library to the class path. Right-click the Scheduler project and select Project Properties > Libraries and Classpath > Add Library > Oracle Enterprise Scheduler Extensions.
19. Deploy the libraries `oracle.xdo.runtime` and `oracle.xdo.webapp` to the Oracle Enterprise Scheduler UI managed server. These libraries are located in the directory `${MW_HOME}/jdeveloper/xdo`, where `${MW_HOME}` is the Oracle Fusion Middleware home directory.

20. Deploy the application.

---

**Note:**

When testing the UI in a web browser, you may need to add a security exception to your browser so that the UI renders correctly. Follow the directions in the online help for your web browser.

---

**Example 65-34 Passing Values in a Map Object to a Java Job**

```java
public void execute(RequestExecutionContext ctx, RequestParameters props) throws ExecutionErrorException, ExecutionWarningException, ExecutionCancelledException, ExecutionPausedException {
    String pageTitle = (String) props.getValue("pageTitle");
    // Retrieve other parameters.
    // ...
}
```

**Example 65-35 Creating a View Object Using a Query**

```sql
select null as Attribute1, null as Attribute2 from dual
```

65.15.2 How to Add a Custom Task Flow to an Oracle ADF User Interface for Submitting Job Requests

You can add a custom task flow to an Oracle ADF user interface used to submit job requests at runtime.

To add a custom task flow, do the following:

1. Create a task flow and bind it to your Oracle ADF user interface for submitting a job request created in How to Create an User Interface for Submitting Job Requests.

2. Create an ADF Business Components view object for each UI field. Name the view objects that are bound to UI fields `ParameterVO1`, `ParameterVO2`, and so on.

   Name the attributes of the view objects as follows: `ATTRIBUTE1`, `ATTRIBUTE2`, and so on.

3. Include the view objects in the relevant application module. Even if their names are different, the view object instance names ought to be `ParameterVO1`, `ParameterVO2`, `ParameterVO3`, and so on.

4. In the job definition, define the properties `CustomDataControl` and `ParameterTaskflow`. For more information, see Understanding Job Definitions and Job Types.

5. Optionally, include the method `preSubmit()` in the application module. Oracle Enterprise Scheduler invokes this method before retrieving the parameter values for the submission request.

   Your implementation of the `preSubmit()` method (which returns a Boolean value) could include validation code in the custom task flow. If the validation fails,
your code can throw an exception with the proper internationalized error message. If this validation fails while submitting the request, the error message is displayed to the user and the submission doesn’t go through.

65.15.3 About Handling Schedule Request Submission and Jobset UI Submit and Cancel Button Events

The schedule request submission and jobset UI taskflows include Submit and Cancel buttons. Typically, clicking the Submit button submits the job request, closes the transaction, and returns the user to the page that launched the schedule request submission or jobset UI taskflow (the container taskflow). Clicking the Cancel button resets the internal data structures in the schedule request submission or jobset UI and returns the user to the page that launched the schedule request submission or jobset UI taskflow.

Clicking the Submit or Cancel buttons notifies the containing bounded or unbounded parent taskflow of the result of the Submit or Cancel event, and the container taskflow decides what to do next. The containing taskflow may be a popup or an inline page. The container taskflow handles any navigation required after clicking either button.

The schedule request submission and jobset UI taskflows define two root outcomes and two parent outcomes, which the parent taskflow can use to handle navigational requirements.

The following sample shows the schedule request submission UI parent outcomes.

The following sample shows the jobset UI parent outcomes.

As shown in the preceding samples, the containing taskflow defines the root/parent outcomes that occur when the user clicks the Submit or Cancel button, respectively. These outcomes are onSRSSubmitted and onSRSCanceled, in the case of the schedule request submission UI taskflow, and onJobsetRequestSubmitted and onJobsetRequestCanceled in the case of the jobset UI taskflow.

The consuming or parent taskflow uses these root/parent outcomes in their view definition files (*taskflow.xml or adfc-config.xml) and defines control flow rules accordingly.

Note:
The taskflows must be dropped as a region on a page for the parent actions to work. Make sure to drop the schedule request submission or jobset UI taskflow as a region on a page.

Example 65-36 Schedule request submission UI parent outcomes

```xml
<parent-action id="rootSubmitActionId">
  <description id="__1">Parent action when the submit button is clicked for root parent</description>
  <root-outcome id="submitOutcome">onSRSSubmitted</root-outcome>
</parent-action>

<parent-action id="rootCancelActionId">
  <description id="__2">Parent action when the cancel button is clicked for root parent</description>
  <root-outcome id="cancelOutcome">onSRSCanceled</root-outcome>
</parent-action>
```
Example 65-37  Jobset UI parent outcomes

65.15.3.1 About Handling Schedule Request Submission and Jobset UI Submit and Cancel Button Events

In some cases, you may want to place the taskflow in a region component within an explicitly defined popup window. In such cases, the taskflow must be refreshed after the popup window closes.

65.15.3.1.1 How to Display the Popup Window

To display the popup window, do the following:

1. Drop the taskflow onto the popup window as a dynamic region. Dynamic regions have the advantage of not being tied to a single taskflow ID.

2. In your page definition file, map the taskflow’s taskflowId attribute to a method in your managed bean. This bean should typically be in a view scope.

   The managed bean method should return the taskflow ID as a string. By default, the ID should be an empty string, but not null. You can define a string member variable to store the current taskflow ID and return that string variable to the method.
3. Define a `popupFetchListener` in your JSFF file, and map it to a method in the same managed bean. In this `popupFetch` method, you can swap the taskflow ID with the schedule request submission taskflow ID, so that the method returning the taskflow ID now returns the schedule request submission taskflow ID.

When the popup window opens, it displays the schedule request submission UI taskflow.

### 65.15.3.1.2 How to Close the Popup Window

To close the popup window, do the following:

1. In the view definition file of your taskflow, define the control flow rules to handle the parent actions to be returned by the schedule request submission UI taskflow upon clicking the Submit and Cancel buttons.

2. Define a method call activity in the same taskflow definition file, and call that method from the control flow rule. This method call activity can call a method in the same view scope managed bean.

3. The managed bean method should swap the schedule request submission UI taskflow ID with an empty taskflow ID, an empty string. The taskflow ID in the page definition file (and the dynamic region) will now reference an empty taskflow.

The taskflow is now finalized, and garbage collection has occurred. The schedule request submission taskflow has been re-initialized. The next time the schedule request submission UI taskflow is called, it will execute a new instance of the taskflow.

### 65.15.3.2 How to Handle Schedule Request Submission and Jobset UI Submit and Cancel Buttons in an ADF Popup Window

Following is an example of how the parent taskflow can use these parent/root outcomes to define the navigational rules for closing or navigating away from the schedule request submission UI taskflow. The steps are the same for jobset UI taskflows, but the outcomes have different names.

In this example, the schedule request submission UI taskflow is placed in an ADF popup window. When the user clicks the Submit or Cancel button, the popup window closes, returning the user to the page that launched the popup.

**Note:**

This is only an example of how to use the root/parent outcomes. The actual implementation may vary according to the usecase of the user.

You can either use the parent or root outcome at any given time, depending on the use case. By default, the schedule request submission or jobset UI taskflows always pass a parent outcome. If the consuming application needs a root outcome, then you must pass the `requireRootOutcome` parameter with a value of `true` to the schedule request submission or jobset UI taskflow.

To handle **Submit** and **Cancel** buttons, do the following:

1. Define the control flow rule in your view definition file.
<control-flow-rule>
  <from-activity-id>*</from-activity-id>
  <control-flow-case id="__101">
    <from-outcome id="__121">onSRSCanceled</from-outcome>
    <to-activity-id id="__111">cancelSRS</to-activity-id>
  </control-flow-case>
  <control-flow-case id="__102">
    <from-outcome id="__122">onSRSSubmitted</from-outcome>
    <to-activity-id id="__112">cancelSRS</to-activity-id>
  </control-flow-case>
</control-flow-rule>

2. Define the taskflow activity to handle the navigation. This activity is invoked by the control flow rule. A method call activity has been defined in this example.

<method-call id="cancelSRS">
  <method>#{myManagedBean.closeSRSPopup} </method>
  <outcome>
    <fixed-outcome>srsPopupClosed</fixed-outcome>
  </outcome>
</method-call>

3. Define the corresponding method in the managed bean of the parent taskflow.

```java
public void closeSRSPopup() {
    RichPopup srspopup = getPopup();
    if (srspopup != null) {
        srspopup.cancel(); // closes the popup
    }
}
```

4. To use a root outcome, set the `requireRootOutcome` taskflow parameter to true.

5. Deploy the parent taskflow and launch the page to test out the navigation.

65.15.3.3 About Schedule Request Submission and Jobset UI Submit and Cancel Buttons in a UIShell Popup Window

If you are launching the schedule request submission taskflow in a UIShell popup window, then use the `closePopup()` method described in Implementing OK and Cancel Buttons in a Popup.

If you are launching the schedule request submission UI taskflow in the main task area of the UIShell, then you need to follow the navigation options described in How to Implement End User Preferences.

---

**Note:**

The UIShell APIs `closeMainTask()` and `openMainTask()` can only be invoked from within a bounded taskflow. You must wrap the schedule request submission UI taskflow in a dummy container taskflow, and define the control flow rules to consume the parent actions in the view definition file of the container taskflow.
65.15.4 How to Enable Support for Context-Sensitive Parameters in an Oracle ADF User Interface for Submitting Job Requests

After integrating your application with the Oracle ADF UI for submitting job requests, enable context-sensitive parameter support in the UI.

The request submission UI will render the context-sensitive parameters first so that the end user will specify the context-sensitive parameter values. Context is set in the database based on these parameters. After setting the context, it renders the rest of the parameters based on context set at database layer. When the job runs, the actual business logic will run after setting the context based on the context-sensitive parameter values inside the database.

To enable context-sensitive parameter support in the UI, do the following:

1. Follow the instructions described in How to Create an User Interface for Submitting Job Requests.
2. Create a native ADF Business Components view object with attributes CTXATTRIBUTE1, CTXATTRIBUTE2, and so on, with a maximum of 100 attributes.
   For example, create a view object with the query Select null as CTXATTRIBUTE1, CTXATTRIBUTE2, CTXATTRIBUTE3 from dual. Include required UI hints such as display label, tool tip, and so on.
3. Create a PL/SQL procedure or function to set the context.
4. Specify the parameters shown in Example 65-38 and Example 65-39 in the job definition metadata.
   - contextParametersVO: Enter the fully qualified name of the view object that holds the context sensitive parameters.
   - setContextAPI: PL/SQL API to set the context, along with the package name. The _myPkg1.mySetCtx procedure receives arguments based on attributes in the contextParametersVO.

Example 65-38 contextParametersVO
<parameter name="contextParametersVO" data-type="string">_oracle.apps.mypkg.TestCtxVO</parameter>

Example 65-39 setContextAPI
<parameter name="setContextAPI" data-type="string">_myPkg1.mySetCtx</parameter>

65.15.5 How to Save and Schedule a Job Request Using an Oracle ADF UI

Saving and scheduling a job request using an Oracle ADF UI involves using the Oracle Enterprise Scheduler Extensions library with a JSF application that includes a task flow in which a job is scheduled and saved.

To schedule a job request, do the following:

1. Follow the instructions in How to Create an User Interface forSubmitting Job Requests up to step 9.
Note:
If the custom parameters task flow has no transactions of its own, it must set the `data-control-scope` to `isolated`. This ensures that multiple `parametersVO` properties using the same application module get their independent application module instance.

2. Drag and drop the `SaveSchedule-taskflow` object onto the dialog. No input parameters are required.

3. When prompted, add the required library to the ViewController project by clicking Add Library. Save the JSF page.

4. In the JSF application you created, create another project called Scheduler. Select File > New, and choose General > Empty Project. This project will be used to create Oracle Enterprise Scheduler metadata and job implementations.

5. In the Scheduler project, add the Oracle Enterprise Scheduler Extensions library to the class path. Right-click the Scheduler project and select Project Properties > Libraries and Classpath > Add Library > Oracle Enterprise Scheduler Extensions.

6. Deploy the application.

7. Start the application using the following URL:


8. Enter a schedule name, description and package name with the namespace appended, as shown in Figure 65-8.

   **Figure 65-8  Saving a Job Submission Schedule**

   ![Image of schedule creation interface]

9. Save the schedule.

   A message displays indicating the metadata object ID of the saved schedule. This ID can be used for further job or job set request submissions.
65.15.6 How to Submit a Job Using a Saved Schedule in an Oracle ADF UI

Submitting a saved job request schedule using an Oracle ADF UI involves using the Oracle Enterprise Scheduler Extensions library with a JSF application that includes a task flow in which a saved job schedule can be submitted.

To submit a job, do the following:

1. Follow the instructions in How to Create an User Interface for Submitting Job Requests.

2. Deploy the application. Open the page using the following URL:

3. Click the Schedule tab. In the Run option field, select the Use a Schedule radio button.

4. From the Frequency drop-down list, select Use a Saved Schedule.

5. Enter the namespace and package names for the schedule along with the name of the schedule.

6. To view the list of scheduled jobs, click Get Details. Click Submit to submit the saved job request.

65.15.7 About Notifying Users of the Status of Executed Jobs

The Oracle ADF user interface for submitting job requests provides the ability to notify users of the status of submitted jobs (via the Notification tab of the user interface). For example, users can request a notification to be sent to the originator of the job request.

A notification includes two components: the user to whom the notification is to be delivered, and the completion status of the job that triggers the notification. For example, notifications can be sent upon the successful completion of a job, or when a job completes in an error or warning state.

65.15.8 What Happens When You Create an Oracle ADF User Interface for Submitting Job Requests

The Oracle ADF interface is integrated with the Oracle Fusion application, and the application is tested and deployed. End users access the Oracle ADF user interface, fill in optional job properties, and click a button to submit the job request.

65.15.9 What Happens at Runtime: How an Oracle ADF User Interface for Submitting Job Requests Is Created

The application receives the submitted job request and calls Oracle Enterprise Scheduler to run the job. The Oracle Fusion application accesses the values of the properties entered by end users through the view object in which these properties were defined at design time. The job returns a result of success or failure, and the result passes from the Oracle Fusion application to Oracle Enterprise Scheduler.

Custom Task Flow

A job that includes properties to be filled in by end users through an Oracle ADF user interface at runtime includes ADF Business Components view objects with validation.
and the parameters to be filled in by end users. These parameters are submitted at runtime in the order in which they have been defined, meaning the first custom parameter to be defined is submitted first. The custom parameters must be named as follows:

ParameterVO1.ATTRIBUTE1, ParameterVO1.ATTRIBUTE2, ParameterVO2.ATTRIBUTE1, ParameterVO3.ATTRIBUTE1, and so on.

If the job definition includes the properties ContextParametersVO, ParameterTaskflow and parametersVO, these properties render in that order at runtime.

Context-Sensitive Parameters

When starting the job request submission page UI to submit a job or job set request with context-sensitive parameters, the contextParametersVO parameter initially renders in the Parameters tab of the Oracle ADF user interface. The end-user can then enter values for the context-sensitive parameters. Clicking Next invokes an API called setConextAPI by passing the context parameters. The context is set at the database level and the remaining parametersVO job parameters are rendered.

When the context-sensitive parameters are modified, end users must click Next to set the context with the new values.

Notifications

When the final status of the job is determined, Oracle Enterprise Scheduler delivers the notifications to the relevant users using the User Messaging Service. Users receive notifications based on their messages preferences. The notification view object defined at design time populates the input box in the submission request user interface at runtime.

65.16 Specify Optional Parameters

TBD

* Applying Language Templates
* Specifying a Printer for Language Output
* Configuring Automatic Notifications Upon Completion of Language-Specific Requests

65.17 About Submitting Job Requests Using the Request Submission API

You can submit, cancel and otherwise manage job requests using the request submission API.

For information about using the request submission API, see the chapter "Using the Runtime Service" in Developing Applications for Oracle Enterprise Scheduler.

65.17.1 About Using the Java Request Submission API

Use the Oracle Enterprise Scheduler Java API to request job submission in a Java application. For more information about the API, see the Oracle Enterprise Scheduler Service javadoc documentation.

Sample code: TBD
65.17.2 About Using the PL/SQL Request Submission API

The package ESS_RUNTIME_SERVICE enables submitting job requests using PL/SQL. The following samples illustrate the use of the PL/SQL request submission API.

In the first example, a request is submitted to run a job defined in the metadata of a J2EE application called app1. The procedure takes two arguments set to myarg1 and myarg2. The job is scheduled to run every two hours beginning at the time of job submission.

In the second example, a subrequest is submitted. A sub request is a request submitted from a running job, where the running job passes its own execution context in the form of a parameter called request_handle. Passing the execution context ties the request being submitted to the currently running request. This routine is designed to be called from a running PL/SQL request.

The request is submitted to J2EE application called app1, and the procedure takes two arguments whose values are myarg1 and myarg2. The job is scheduled to run every two hours from the time of submission.

**Example 65-40  Submitting a job request using the PL/SQL API**

request_prop_table_t myprops;
ess_runtime.set_submit_args( myprops, 'myarg1', 'myarg2');
ess_runtime.submit_request_adhoc_sched(
    application => 'app1',
    definition_type => 'JOB',
    definition_name => 'jobA',
    interval=>2,
    frequency=>'HOURLY',
    props => myprops);

**Example 65-41  Submitting a Subrequest Using the PL/SQL API**

procedure myjob (request_handle IN varchar2, arg1 IN varchar2, arg2 IN varchar2) as
    number reqid;
    request_prop_table_t myprops;
    ...
    begin
        ...
        reqid := ess_runtime.get_request_id(request_handle);
        ess_runtime.set_submit_args( myprops, 'subarg1', 'subarg2');
        ess_runtime.submit_subrequest(
            request_handle => request_handle,
            definition_name => 'jobA',
            definition_package => 'Test_Package',
            props => myprops)
        ...
    end

65.17.3 About Using the C Request Submission API

C routines are provided to enable request submission. For more information, see TBD. Currently, no samples are available.
65.18 About Defining Oracle Business Intelligence Publisher Postprocessing Actions for a Scheduled Job

Oracle Business Intelligence Publisher enables generating reports from a variety of data sources, such as Oracle Database, web services, RSS feeds, files, and so on. BI Publisher provides several delivery options for generated reports, including print, fax, and email.

To create an Oracle BI Publisher report, an Oracle BI Publisher report definition is required. Oracle BI Publisher report definitions consist of a data model that specifies the type of data source (database, web service, and so on) and a template for output formatting.

With report definitions in place, options for reporting are available to end users in the Output tab of the Oracle ADF user interface. The Output tab provides options through which an end user can define templates for reports. They can specify layout templates, document formats (such as PDF, RTF, and more), report destinations (email addresses, fax numbers, or printer addresses), and so on. When the user submits a request, this information is stored in the Oracle Enterprise Scheduler schema. The preprocessor then invokes the Oracle BI Publisher service and passes the saved data to it.

Extensions to Oracle Enterprise Scheduler provide the ability to run Oracle BI Publisher reports as batch jobs. The Oracle Enterprise Scheduler postprocessing infrastructure enables applying Oracle BI Publisher formatting templates to XML data and delivering the formatted reports by printing, faxing, and so on.

Related Links

The following documents provide additional information related to subjects discussed in this section:

- For more information about defining postprocessing actions for scheduled jobs, see "Creating a Business Domain Layer Using Entity Objects" in the Developing Fusion Web Applications with Oracle Application Development Framework.

- For more information on the web service, see the chapter "Using the Oracle Enterprise Scheduler Web Service" in Developing Applications for Oracle Enterprise Scheduler.

- For more information about configuring security certificates, see the chapter "Managing Keystores, Wallets, and Certificates" in the Administering Oracle Fusion Middleware.

65.18.1 Perform Prerequisite Tasks

Before you start defining Oracle BI Publisher postprocessing for a scheduled job, do the following:

1. Follow the instructions for setting up Oracle BI Publisher reporting as described in the "Creating and Editing Reports" chapter in Report Designer's Guide for Oracle Business Intelligence Publisher.

   Use the following file to set up reporting and seed your database with the relevant Oracle BI Publisher data:

2. Create an Oracle BI Publisher job definition, following the instructions in the Oracle BI Publisher documentation.
3. Define File Management Group (FMG) properties for the Oracle BI Publisher job definition as described in How to Define File Groups for a Job.

Example 65-42 Location of the File for Setting Up Oracle BI Publisher Reporting and Seeding the Database

$BEAHOME/jdeveloper/jdev/oaext/adflib/PPActions.jar

65.18.2 How to Define Oracle BI Publisher Postprocessing for a Scheduled Job

Defining postprocessing for a scheduled job involves the following:

- Define the postprocessing action.
- Create a Java class for the postprocessing action. The Java class uses the parameters collected by the Oracle Enterprise Scheduler UI and calls Oracle BI Publisher APIs as required.
- Create a native ADF Business Components view object to save parameters for postprocessing, such as template name, output format, locale, and so on.

To create an Oracle BI Publisher postprocessing action, do the following:

1. In the table called APPLCP_PP_ACTIONS, define the postprocessing action to be executed for the job.

The columns to be seeded in the APPLCP_PP_ACTIONS table are as follows:

- **Action_SN**: Define a short name for the action, used when postprocessing actions are submitted programatically. For example, OBFUSC8.
- **Action Name**: Enter a name for the action to be displayed in the user interface. This name is stored separately for translation purposes.
- **Class**: Enter the name of the Java class that defines the logic for the postprocessing action. For example, oracle.apps.shh.obfuscate.PPobfuscate.
- **VO_Def_Name**: Enter the name of the view object used to collect the arguments for the postprocessing action. For example, oracle.apps.shh.obfuscate.PPobfuscateVO.
- **Type**: Enter the category of the postprocessing action to be taken. Enter one of the following categories of postprocessing actions:
  - **L**: Indicates a Layout postprocessing action. Layout actions change the output of the job, and produce new output.
  - **O**: Indicates an Output postprocessing action. Output actions act on the output created by the job and its layout actions, performing delivery, publishing, printing, and so on.
  - **F**: Indicates a Final postprocessing action. Final Actions take no input. Final postprocessing actions execute using the final status of the job after all Layout and Output actions have executed.
- **On_Success**: Indicate whether the postprocessing action runs following a successful job. Enter Y or N.
• **On Warning**: Indicate whether the postprocessing action runs following a job that ends in a warning. Enter Y or N.

• **On Failure**: Indicate whether the postprocessing action runs following a failed job. Enter Y or N.

• **SEQ_NUM**: Enter a number to sequentially order the postprocessing actions. Only registered postprocessing actions of the same type can be sequentially ordered. This value determines both the order in which the tabs corresponding to the actions appear in the user interface, and the order in which the actions run.

Each action can also specify request parameters used by the postprocessing action view object. These parameters must be set in the job definition for any job using this action. The parameter names are stored in the APPLCP_PP_ACTION_PARAMS table. The values of these parameters are accessible from the parameter view object at the time of job request submission. Postprocessing actions can access all request parameters at runtime using the request ID.

2. Define a Java class for the postprocessing action, implementing the interface oracle.apps.fnd.applcp.request.postprocess.PostProcess. Use the methods required by the interface as described in Table 65-4.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>invokePostProcess(long requestID, String ppArguments[], ArrayList files);</td>
<td>Receives the requestID parameter, the ppArguments[] array of arguments collected from the view object (or submitted programmatically), and the files array list which identifies the files on which the action is to be taken. It is possible to specify the location of the output file.</td>
</tr>
<tr>
<td>ArrayList getOutputFileList();</td>
<td>Returns an array of the output files created by the postprocessing action.</td>
</tr>
</tbody>
</table>

Additional methods used by the invokePostProcess method are shown in Table 65-5.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>runReport ()</td>
<td>Enables the postprocessing action to pass to the Business Intelligence Publisher the job’s XML output along with the template ID and format (all collected during job request submission).</td>
</tr>
</tbody>
</table>

Additional methods used by the ReportRequest object are shown in Table 65-6.
Table 65-6 Oracle BI Publisher Client API
oracle.xdo.service.client.types.ReportRequest Used by the ReportRequest Object

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setAttributeFormat()</td>
<td>Set the format for the Oracle BI Publisher report request.</td>
</tr>
<tr>
<td>setAttributeLocale()</td>
<td>Set the locale data for the Oracle BI Publisher report request.</td>
</tr>
<tr>
<td>setAttributeTemplate()</td>
<td>Set the template for the Oracle BI Publisher report request.</td>
</tr>
<tr>
<td>setXMLData()</td>
<td>Set the XML data for the Oracle BI Publisher report request.</td>
</tr>
</tbody>
</table>

An example of a Java class that defines a postprocessing action is shown in Example 65-43:

3. Create a native ADF Business Components view object to collect the parameters to be used in the postprocessing action. Follow the procedure described in About Creating a Job Definition. Define any view object attributes sequentially.

If the view object requires access to action-specific values from the job definition, specify the required job definition parameters in the action definition. The submission UI automatically retrieves the values from the job definition metadata and sets them as Oracle Fusion Middleware Extensions for Applications (Applications Core) Session attributes that may be retrieved using the ApplSession standard API.

Example 65-43 A Java Class that Defines a Postprocessing Action

```java
package oracle.apps.shh.Obfuscate;

import oracle.apps.fnd.applcp.request.postprocess.PostProcess;
import oracle.apps.fnd.applcp.util.ESSContext;
import oracle.apps.fnd.applcp.util.PostProcessState;
import oracle.as.scheduler.*;

public class PPobfuscate implements PostProcess {
    ArrayList myOutputFiles;

    ArrayList getOutputFileList() {
        return myOutputFiles;
    }

    PostProcessState invokePostProcess(long requestID, String ppArguments[],
                                        ArrayList files)
    {
        RuntimeService rService = null;
        RuntimeServiceHandle rHandle = null;
        try {
            // Accessing Runtime Details for a given requestID
            RequestDetail rDetail = null;
            RequestParameters rParam = null;
            String obfuscationSeed = ppArguments[0];
            String codedFileName = ppArguments[1];
```
String myNewFile;
String outDir = null;

rService = ESSContext.getRuntimeService();
if (rService != null) rHandle = rService.open();
if (rHandle != null) rDetail = getRequestDetails(rHandle, requestID);
if (rDetail != null) rParam = rDetail.getParameters();
if (rParam != null) outDir = rParam.getValue("outputWorkDirectory");
if (outDir == null) 
    // Details not received, usually an exception would have been thrown
    // by now. We handle this case to be robust.
    // Log the ERROR to Oracle Diagnostic Logging
    return PostProcessState.ERROR;
}
// Check files
if (files[0] == null) 
    // no files - PostProcessing should never call us in this state
    // in case it does - log Error to Oracle Diagnostic Logging
    return PostProcessState.ERROR;

// This example expects a single file
myNewFile = outputDir + System.getProperty("file.separator") +
codedFileName;
Obfuscate.performObfuscation( files[0], obfuscationSeed, myNewFile );
myOutputFiles[0] = myNewFile;

// In case multiple files are used.
for (i = 1; files[i] != null; i++) 
    // Appending a counter to the filename to be unique.
    myNewFile = outputDir + System.getProperty("file.separator") +
codedFileName + i;
    Obfuscate.performObfuscation( files[i], obfuscationSeed, myNewFile );
    myOutputFiles[i] = myNewFile;
}
return PostProcessState.SUCCESS;
}

} // end class

65.18.3 About Defining Oracle BI Publisher Postprocessing Actions for a Scheduled PL/SQL Job

Example 65-44 shows a PL/SQL job that includes Oracle BI Publisher postprocessing actions. The PL/SQL job calls the method ess_runtime.add_pp_action so as to
generate a layout for the data from the postprocessing action. This example formats
the XML generated by the job as a PDF file.

**Example 65-44  Defining a Scheduled PL/SQL Job with Oracle BI Publisher
Postprocessing Actions**

declare
l_reqid   number;
l_props   ess_runtime.request_prop_table_t;
begin
.
   ess_runtime.add_pp_action ( 
      props              => l_props,               -- IN OUT
      request_prop_table_t,
      action_order       => 1,                     -- order in which this post
processing action will execute.
      action_name        => 'BIPDocGen',           -- Action for Document
      Generation (layout)
      on_success         => 'Y',                   -- Should this be called on
success,
      on_warning         => 'N',                   -- Should this be called on
warning,
      on_error           => 'N',                   -- Should this be called on
error,
      file_mgmt_group    => 'XML',                 -- File types this action
will process. It has to be defined in Job Defintion,
      step_path          => NULL,                  -- IN varchar2 default NULL,
      argument1          => 'XLABIPTEST_RTF',      -- Template name needed for
Documnet Generation action,
      argument2          => 'pdf'                  -- What type of layout file
will be generated by Document Generation action,
    );
    l_reqid :=
      ess_runtime.submit_request_adhoc_sched 
      (application => 'SSEssWls',               -- Application
      Application
      definition_type => 'JOB',
      definition_name => 'BIPTestJob',
      definition_package => '/mypackage',
      props => l_props);
commit;
dbms_output.put_line('request_id = :'||l_reqid);
end;

**65.18.4 About Invoking Postprocessing Actions Programmatically**

You can invoke postprocessing actions programmatically from a client using a Java or
web service API. Both APIs require the same set of parameter values described in table
Table 65-7.

For Java clients, call the `addPPAction` method of
`oracle.as.scheduler.cp.SubmissionUtil`. The method takes the values
needed to invoke the action and throws an exception called
`IllegalArgumentException` if the number of arguments exceeds 10. **Example 65-45** shows the declaration of the method.

For web service clients, you invoke the method using a proxy, as in **Example 65-46**.
### Table 65-7  Parameters for Adding a Postprocessing Action

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>params</td>
<td>A RequestParameters object into which this method adds parameters.</td>
</tr>
<tr>
<td>actionOrder</td>
<td>The ordinal location of this action in the sequence of actions to be performed within the action domain. Oracle BI Publisher process requests starting with action order index 1.</td>
</tr>
<tr>
<td>actionName</td>
<td>The name of the action to perform. The following lists acceptable values for this parameter, along with the acceptable values you can use in the arguments parameter of this method.</td>
</tr>
<tr>
<td></td>
<td>• BIPDocGen: for applying Oracle BI Publisher templates. Acceptable argument parameter values are:</td>
</tr>
<tr>
<td></td>
<td>– argument1: maps to report parameter TEMPLATE, the template name.</td>
</tr>
<tr>
<td></td>
<td>– argument2: maps to report parameter OUTPUT_FORMAT, the output format for Oracle BI Publisher document generation, for example, &quot;pdf&quot; or &quot;html&quot;.</td>
</tr>
<tr>
<td></td>
<td>– argument3: maps to report parameter LOCALE, the locale to be used while generating output.</td>
</tr>
<tr>
<td></td>
<td>• BIPPrintService: for specifying the print action. Acceptable argument parameter values are:</td>
</tr>
<tr>
<td></td>
<td>– argument1: maps to printerName</td>
</tr>
<tr>
<td></td>
<td>– argument2: maps to numberOfCopies</td>
</tr>
<tr>
<td></td>
<td>– argument3: maps to side</td>
</tr>
<tr>
<td></td>
<td>– argument4: maps to tray</td>
</tr>
<tr>
<td></td>
<td>– argument5: maps to pagesRange</td>
</tr>
<tr>
<td></td>
<td>– argument6: maps to orientation</td>
</tr>
<tr>
<td></td>
<td>• BIPDeliveryEmail: for specifying the email action. Acceptable argument parameter values are:</td>
</tr>
<tr>
<td></td>
<td>– argument1: maps to emailServerName</td>
</tr>
<tr>
<td></td>
<td>– argument2: maps to from</td>
</tr>
<tr>
<td></td>
<td>– argument3: maps to to</td>
</tr>
<tr>
<td></td>
<td>– argument4: maps to cc</td>
</tr>
<tr>
<td></td>
<td>– argument5: maps to bcc</td>
</tr>
<tr>
<td></td>
<td>– argument6: maps to replyTo</td>
</tr>
<tr>
<td></td>
<td>– argument7: maps to subject</td>
</tr>
<tr>
<td></td>
<td>– argument8: maps to messageBody</td>
</tr>
<tr>
<td></td>
<td>• BIPDeliveryFax: for specifying the fax action. Acceptable argument parameter values are:</td>
</tr>
<tr>
<td></td>
<td>– argument1: maps to faxServerName</td>
</tr>
<tr>
<td></td>
<td>– argument2: maps to faxNumber</td>
</tr>
<tr>
<td>description</td>
<td>Description of this post processor action.</td>
</tr>
<tr>
<td>onSuccess</td>
<td>Determines whether this action should be performed on successful completion of the job.</td>
</tr>
<tr>
<td>onWarning</td>
<td>Determines whether this action should be performed when the job or step has completed with a warning.</td>
</tr>
<tr>
<td>onError</td>
<td>Determines whether this action should be performed when the job or step has completed with an error.</td>
</tr>
</tbody>
</table>
### Table 65-7 (Cont.) Parameters for Adding a Postprocessing Action

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileMgmtGroup</td>
<td>The name of the File Management Group. When using a Oracle BI Publisher template, the value of this parameter is XML, as defined in the job definition Program.FMG property with the value L.XML.</td>
</tr>
<tr>
<td>arguments</td>
<td>A list of arguments for the post processor action. See the actionName parameter for values you can use for the arguments parameter.</td>
</tr>
</tbody>
</table>

#### Example 65-45 Sample declaration of the addPPAction method

```java
public static void addPPAction (RequestParameters params, 
    int actionOrder, 
    String actionName, 
    String description, 
    boolean onSuccess, 
    boolean onWarning, 
    boolean onError, 
    String fileMgmtGroup, 
    String[] arguments) 
    throws IllegalArgumentException
```

#### Example 65-46 Adding Postprocessing Actions for a Request

```java
ESSWebService proxy = createProxy("addPPActions");

PostProcessAction ppAction = new PostProcessAction();
ppAction.setActionOrder(1);
ppAction.setActionName("BIPDocGen");
ppAction.setOnSuccess(true);
ppAction.setOnWarning(false);
ppAction.setOnError(false);
ppAction.getArguments().add("argument1");
ppAction.getArguments().add("argument2");

List<PostProcessAction> ppActionList = new ArrayList<PostProcessAction>();
ppActionList.add(ppAction);

RequestParameters reqParams = new RequestParameters();
reqParams = proxy.addPPActions(reqParams, ppActionList);
```

### 65.18.5 What Happens When You Define Oracle BI Publisher Postprocessing Actions for a Scheduled Job

Depending on the FMG property set for the job definition, the relevant postprocessing action is selected for the job.

The ppArguments array stores the values collected from the view object attributes. The array is passed to the `invokePostProcess` method which executes in the Java class that defines the postprocessing action.
65.18.6 What Happens at Runtime: How Oracle BI Publisher Postprocessing Actions are Defined for a Scheduled Job

At runtime, the user interface uses the view object to collect the arguments for executing the postprocessing action as defined in the table `APPLCP_PP_ACTIONS`. These arguments also instruct the user interface as to how to invoke the action logic.

The postprocessing action accesses the XML output file from the job request, and passes the XML output to Oracle BI Publisher. The postprocessing action creates a report request containing the XML data.

The postprocessing action displays in the submission Oracle ADF UI. The UI enables adding a postprocessing action for the scheduled job, selecting arguments for the action using the view object and selecting output options for the action. The user interface also displays the name of the File Management Group with which the output files are associated.

Note:

When testing the UI in a web browser, you may need to add a security exception to your browser so that the UI renders correctly. Follow the directions in the online help for your web browser.

65.19 About Monitoring Scheduled Job Requests Using an Oracle ADF UI

It is possible to view previously submitted jobs by integrating the Monitoring Processes task flow into an application.

For information about enabling tracing for jobs, see Debugging and .

Related Links

The following document provides additional information related to subjects discussed in this section:

For more information about tracing Oracle Enterprise Scheduler jobs, see the section "Tracing Oracle Enterprise Scheduler Jobs" in the chapter "Managing Oracle Enterprise Scheduler Service and Jobs" in the Administrator’s Guide.

65.19.1 How to Monitor Scheduled Job Requests

The main steps involved in monitoring scheduled job requests using an Oracle ADF UI are as follows:

- Configure Oracle Enterprise Scheduler in JDeveloper
- Create and initialize an Oracle Fusion web application
- Create a UI Shell page and drop the Monitor Processes task flow onto it

Note:

Fields such as submission date, ready time, scheduled date, process start, name, type, definition, and so on, are not set unless the job request or subrequest is successfully validated.
To monitor scheduled job requests, do the following:

1. Follow the instructions in How to Create an User Interface for Submitting Job Requests up to and including step 5.

2. Under the ViewController project, right-click Web Content and create a new JSF page called Consumer.jspx. Select the following options:
   - UIShell (template)
   - Create as XML Document

3. Create a new JSF page fragment. This page initializes the project.


5. Right-click adfc-config.xml and select Create ADF Menu.
   The Create ADF Menu Model window displays.

6. Rename the default file root_menu.xml to something else.

7. Open the XML file created in the previous step. Look for an itemNode element as follows:

   `<itemNode id="itemNode_JSF/JSPX page name">`

   For example, the Consumer.jspx page has the following itemNode value:

   `<itemNode id="itemNode_Consumer">`

8. In the Structure window, right-click the root itemNode and select Insert inside itemNode-itemNode_JSF/JSPX page name > itemNode.

9. In Common Properties, enter the following values:
   - `id`: MonitorNode
   - `focusViewId`: /Consumer

10. In Advanced Properties, enter Monitor Processes in the label field.

11. Right-click the itemNode you just added and select Go to Properties.

12. In the Property Inspector, select Advanced and do the following:
   - Select the dynamicMain task type.
   - In the taskFlowId field, enter the following:
     
     ```
     /WEB-INF/oracle/apps/fnd/appclp/monitor/ui/flow/MonitorProcessesMainAreaFlow.xml#MonitorProcessesMainAreaFlow
     ```
   - Enter a string for the pageTitle parameter, which will become the title for the monitoring page. If this parameter is not specified, then the page title will be shown as "Manage Scheduled Processes".

13. Repeat steps 8-12 to create a second itemNode element with the following properties:
   - `id`: __Launcher_itemNode__FndTaskList
• **focusViewId**: /Launcher
• **label**: #{applcoreBundle.TASKS}
• **Task Type**: defaultRegional
• **taskFlowId**: /WEB-INF/oracle/apps/fnd/applcore/patterns/uishell/ui/publicFlow/TasksList.xml#TasksList

14. Right-click adfc-config.xml and select Link ADF Menu to Navigator.


16. Deploy and test the application.

### 65.19.2 How to Embed a Table of Search Results as a Region on a Page

You can embed a table of job request search results as a region on a page. Task flow parameters can be used to further specify the job requests returned by the search.

To embed a search-results table, do the following:

1. Add the **Applications Concurrent Processing (View Controller)** library to the ViewController project.
   
   For more information about adding this library to the project, see *How to Create and Implement a Scheduled Job in JDeveloper*.

2. In the Resource Palette, select **File System > Applications Core > MonitorProcesses-View.jar > ADF Task Flows**.

3. Drag and drop onto the page as a region the **SearchResultsFlow** task flow.

The task flow accepts the following parameters:

• **processId**: The request ID number uniquely identifying the process.

• **processName**: The name of the process, which corresponds to the name of the job definition.

• **processNameList**: Fetches the job requests of multiple process names using a list which contains the relevant job names.

   When specifying the task flow parameter **processName**, this parameter takes precedence over the task flow parameter **processNameList**. The requests returned are for the single process name specified by the **processName** parameter only.

• **scheduledDays**: Queries requests for the last \( n \) days. If this parameter is not specified in a work area task flow, job requests from the last three days are displayed. If the value of this parameter is greater than three days, then the parameter value will be taken as three and only the last three days of job requests display.

• **status**: The status of the request. This filter narrows down the result set to display only the requests with the selected status in the filter.

   If the **status** input parameter is not specified, then the results table shows all requests with all statuses (by default, the **All** value is selected in the status filter list).
If the status input parameter is specified, then the results table show only the requests of the given status. The selected status is chosen as the default in the status filter list.

• isEmbedResults: A Boolean value that indicates whether search results are embedded in the task flow. True or false.
  
  Set to true to embed table results.

• Time Range Filter: This filter is used to narrow down the result set to show only the requests for last \( n \) hours. This filter lists the following values in a dropdown list: (1) Last 1 Hour, (2) Last 12 Hours, (3) Last 24 Hours, (4) Last 48 Hours and (5) Last 72 Hours.
  
The default selected item displays based on the value assigned or given to the task flow parameter scheduledDays.
  
  A scheduledDays value of 1 means the time range filter list displays only the first three items.
  
  A scheduledDays value of 2 means the time range filter list displays only the first four items.
  
  If the value of scheduledDays is 1, then by default, the time range dropdown list displays Last 24 Hours.
  
  If the value of scheduledDays is 3 or more, then by default, the time range dropdown list displays Last 72 Hours.

• pageTitle: When passed, the task flow will render this passed String value as the page title. Optional.

• requireRootOutcome: If the value true is passed, then the task flow will generate a value of root-outcome when the user clicks on the Submit or Cancel buttons. By default the task flow generates a value of parent-outcome.

Specifying more than one of these parameters causes the search to run using the AND conjunction.

65.19.3 How to Log Scheduled Job Requests in an Oracle ADF UI

You can enable Oracle Diagnostic Logging in an Oracle ADF UI used to monitor scheduled job requests. When enabling logging, the UI displays a View Log button.

The View Log functionality in the monitoring UI applies only to scheduled requests with a persistenceMode property set to a value of file. Hence, the View Log button in the scheduled request submission monitoring UI displays only when viewing requests with persistenceMode property set to a value of file.

The only other valid value for the persistenceMode property is the value content. The View Log button is hidden for all requests with a persistenceMode property value of content. If the persistenceMode property is not specified for a given request, then the monitoring UI defaults to a persistenceMode value of file, and displays the View Log button when viewing relevant requests.

To log scheduled job requests, do the following:

1. Open the server's logging.xml file.

2. In the logging.xml file, enter the required logging level for oracle.apps.fnd.applcp.srs, for example: INFO, FINE, FINER or FINEST.
Example 65-47 shows a sample of a logging.xml file with Oracle Diagnostic Logging configured.

3. Save the logging.xml file and restart the server.

Example 65-47 Enabling Logging in the logging.xml File

```xml
<logger name='oracle.apps.fnd.applcp.srs' level='FINEST'
        useParentHandlers='false'>
    <handler name='odl-handler'/>
</logger>
```

65.19.4 Tips for Troubleshooting an Oracle ADF UI Used to Monitor Scheduled Job Requests

Some useful tips for troubleshooting the Oracle ADF UI used to monitor scheduled job requests.

- **Displaying a readable name.** When defining metadata, use the display-name attribute to configure the name to be displayed in the Oracle ADF UI. The monitoring UI will display the value defined for the display-name attribute. If this attribute is not defined, the UI displays the value of the metadata-name attribute assigned to the metadata.

- **Displaying multiple links in the task flow UI that each display a popup window with a different job definition.** The recommended approach is to create a single page fragment that contains the scheduled request submission task flow within an Oracle ADF region. This page is reused by each link to display a different job definition in the scheduled request submission UI. For each link, pass the relevant parameters such as the job definition name, package name, and so on. This approach ensures that the UI session creates and uses a single instance of the task flow.

- **Displaying the correct name given the metadata name and display name attributes.** By default, the display name takes precedence and displays in the UI. If the display name is not defined, then the UI displays the job or job set name.

- **Resolving name conflicts between a job metadata parameter name and a request parameter with the same name.** Oracle Enterprise Scheduler uses the following rules to resolve parameter name conflicts.
  - **The last definition takes precedence.** When the same parameter is defined repeatedly with the read-only flag set to false in all cases, the last parameter definition takes precedence. For example, a property specified at the job request level takes precedence over the same property specified at the job definition level.
  - **The first read-only definition takes precedence.** When the same parameter is defined repeatedly and at least one definition is read-only (that is, the ParameterInfo read-only flag is set to true), the first read-only definition takes precedence. For example a read-only parameter specified at the job type definition level takes precedence over a property with the same name specified at the job definition level, regardless of whether it is read-only.

- **Resolving name conflicts between the job or job set metadata name and display name attributes.** By default, the display name takes precedence over the metadata
name. If the display name is not defined, then the UI defaults to displaying the job or job set name.

- **Understanding the state of a job request.** There are 20 possible states for a job request, each with a corresponding number value. These are shown in Table 65-8.

Table 65-8  Job Request States

<table>
<thead>
<tr>
<th>Job State Number</th>
<th>Job Request State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>UNKNOWN</td>
<td>The state of the job request is unknown.</td>
</tr>
<tr>
<td>1</td>
<td>WAIT</td>
<td>The job request is awaiting dispatch.</td>
</tr>
<tr>
<td>2</td>
<td>READY</td>
<td>The job request has been dispatched and is awaiting processing.</td>
</tr>
<tr>
<td>3</td>
<td>RUNNING</td>
<td>The job request is being processed.</td>
</tr>
<tr>
<td>4</td>
<td>COMPLETED</td>
<td>The job request has completed and postprocessing has commenced.</td>
</tr>
<tr>
<td>5</td>
<td>BLOCKED</td>
<td>The job request is blocked by one or more incompatible job requests.</td>
</tr>
<tr>
<td>6</td>
<td>HOLD</td>
<td>The job request has been explicitly held.</td>
</tr>
<tr>
<td>7</td>
<td>CANCELLING</td>
<td>The job request has been canceled and is awaiting acknowledgement.</td>
</tr>
<tr>
<td>8</td>
<td>EXPIRED</td>
<td>The job request expired before it could be processed.</td>
</tr>
<tr>
<td>9</td>
<td>CANCELLED</td>
<td>The job request was canceled.</td>
</tr>
<tr>
<td>10</td>
<td>ERROR</td>
<td>The job request has run and resulted in an error.</td>
</tr>
<tr>
<td>11</td>
<td>WARNING</td>
<td>The job request has run and resulted in a warning.</td>
</tr>
<tr>
<td>12</td>
<td>SUCCEEDED</td>
<td>The job request has run and completed successfully.</td>
</tr>
<tr>
<td>13</td>
<td>PAUSED</td>
<td>The job request paused for subrequest completion.</td>
</tr>
<tr>
<td>14</td>
<td>PENDING_VALIDATION</td>
<td>The job request has been submitted but has not been validated.</td>
</tr>
<tr>
<td>15</td>
<td>VALIDATION_FAILED</td>
<td>The job request has been submitted, but validation has failed.</td>
</tr>
<tr>
<td>16</td>
<td>SCHEDULE_ENDED</td>
<td>The schedule for the job request has ended, or the job request expiration time specified at submission has been reached.</td>
</tr>
<tr>
<td>17</td>
<td>FINISHED</td>
<td>The job request, and all child job requests, have finished.</td>
</tr>
<tr>
<td>18</td>
<td>ERROR_AUTO_RECOVERY</td>
<td>The job request has run, resulted in an error, and is eligible for automatic retry.</td>
</tr>
<tr>
<td>19</td>
<td>ERROR_MANUAL_RECOVERY</td>
<td>The job request requires manual intervention to be retried or transition to a terminal state.</td>
</tr>
</tbody>
</table>

- **Fixing an Oracle BI Publisher report that does not generate, even though the Oracle Enterprise Scheduler schema REQUEST_PROPERTY table contains all the relevant postprocessing parameters.** Verify that the postprocessing parameters
begin with index value of 1. If a set of parameters begins with an index value of 0 (such as the parameter pp.0.action), then the Oracle BI Publisher report will not generate. Oracle BI Publisher expects parameters to begin with an index value of 1. In the case of a job set with multiple Oracle BI Publisher jobs, verify that all the individual step postprocessing actions begin with an index value of 1.

- Fixing a scheduled request submission UI that does not display, and throws a partial page rendering error in the browser indicating that the drTaskflowId property is invalid. This error may occur due to any of the following.
  - The object oracle.as.scheduler.JobDefinition may be unavailable to the scheduled request submission UI, which attempts to query the object using the MetadataService API.
  - The job definition name or the job definition package name is incorrect when passed as task flow parameters. Ensure that the package name does not end with a trailing forward slash.
  - The metadata permissions are not properly configured for the user who is currently logged in. The JobDefinition object, being stored in Oracle Metadata Services Repository, requires adequate metadata permissions to read and modify the JobDefinition metadata. Ensure that the Oracle Metadata Services Repository to which you are referring contains the job definition name in the proper package hierarchy.

65.20 About Integrating Data Security Task Flows into Applications Setup Manager

Every Oracle application registers task flows with a product called Oracle Fusion Applications Setup Manager. Applications Setup Manager provides a single, unified user interface that allows customers and implementers to configure all Oracle applications by defining custom configuration templates or tasks based on their business needs.

The Applications Setup Manager UI enables customers and implementers to select the business processes or products that they want to implement.

Oracle Function Security controls your privileges to a specific task flow, and users who do not have the required privilege cannot view the task flow. For more information about how to implement function security privileges and roles, see Implementing Function Security.

Table 65-9 lists the task flows and their parameters.

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Parameters Passed</th>
<th>Behavior</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schedule Requests</td>
<td>/WEB-INF/ScheduleRequest-taskflow.xml</td>
<td>None.</td>
<td>Schedules a job request submission.</td>
<td>None.</td>
</tr>
</tbody>
</table>
65.21 About Using a Task Flow Template for Submitting Scheduled Requests Through an Oracle ADF UI

The Oracle ADF UI used to submit scheduled requests supports basic and advanced modes. Switching between modes requires page navigation between two view activities.

In some cases, you may want to use a custom parameter task flow for the UI in the context of an Oracle Fusion web application. One such use case is when you require a method call activity as the default activity of a custom bounded task flow so as to initialize the parameters view object and Flexfield filters defined in that task flow.

When using page navigation between two view activities and custom bounded task flows with a default method call activity, switching between basic and advanced modes might reinitialize the related view objects and entity objects. If this happens, any data entered in basic mode is lost when changing to advanced mode.

The task flow template enables switching between basic and advanced modes in the scheduled request submission Oracle ADF UI without losing data.

Related Links

The following document provides additional information related to subjects discussed in this section:

For more information about creating task flows, see the part "Creating Oracle ADF Task Flows" in Developing Fusion Web Applications with Oracle Application Development Framework.” Alternatively, add the lines of code shown in Example 65-49 to the task flow XML file.

65.21.1 About Using a Task Flow Template for Submitting Scheduled Requests through an Oracle ADF UI

A bundled task flow template is provided, containing the components required to enable switching between basic and advanced modes in the Oracle ADF UI. The task flow template adds a router activity and an input parameter to the custom bounded task flow. Configure the router activity as the default activity.

You need only extend the task flow template as needed and implement the activity IDs defined in the task flow template.

Example 65-48 shows a sample implementation of the task flow template.

The task flow template defines the following:

- A default-activity,
- An input parameter of Boolean type,
- A router activity,
- A control-flow-rule containing two cases.

Example 65-48  Task Flow Template

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<adfc-config xmlns="http://xmlns.oracle.com/adf/controller" version="1.2">
  <task-flow-template id="srs-custom-task-flow-template">
    <default-activity id="defActivity">defaultRouter</default-activity>
    <input-parameter-definition id="param1">
      <!-- Input parameter details here -->
    </input-parameter-definition>
    <router activity="defaultRouter">
      <!-- Router activity details here -->
    </router>
    <control-flow-rule rule="true">
      <case condition="param1=true">
        <!-- Case 1 details here -->
      </case>
      <case condition="param1=false">
        <!-- Case 2 details here -->
      </case>
    </control-flow-rule>
  </task-flow-template>
</adfc-config>
```
65.21.2 How to Extend the Task Flow Template for Submitting Scheduled Requests through an Oracle ADF UI

If you need to create your own custom bounded task flow UI for the parameters section of the scheduled request submission UI, you will need to extend this template. To extend the taskflow template, do the following:

1. When creating a new task flow, extend the task flow by selecting **Use a template**.

   **Note:**
   Ensure your bounded task flow does not define any default activity.

2. Implement the activity IDs defined in the template, which are invoked by the router activity in the template.
   - **initActivity**: The ID of the method call activity.
   - **defaultView**: The ID of the default view activity.

   To do this, to the task flow drag and drop the `createInsert` method from the view object used in the default view. This creates a pagedef file and adds the binding details in `DateBinding.cpx`.

3. Define a control flow rule to navigate from the `initActivity` object to the `defaultView` object. This navigation depends on the outcome of the `initActivity` object, as well as individual use cases.
Example 65-50 shows a sample implementation of a control flow rule.

**Example 65-49  Extending a Task Flow**

```xml
<template-reference>
  <document id="doc1">/WEB-INF/srs-custom-task-flow-template.xml</document>
  <id id="temid">srs-custom-task-flow-template</id>
</template-reference>

**Example 65-50  Implementing a Control Flow Rule**

```xml
<control-flow-rule>
  <from-activity-id>initActivity</from-activity-id>
  <control-flow-case>
    <from-outcome>outcome_of_init_activity</from-outcome>
    <to-activity-id>defaultView</to-activity-id>
  </control-flow-case>
</control-flow-rule>
```

65.21.3 What Happens When you Use a Task Flow Template for Submitting Scheduled Requests through an Oracle ADF UI

Based on the value of the input parameter, the router invokes the method call activity or skips it, and invokes the view activity directly. The Oracle ADF UI must pass the correct parameter values to the task flow while switching modes.

65.21.4 What Happens at Runtime: How a Task Flow Template Is Used to Submit Scheduled Requests through an Oracle ADF UI

When loading the initial page in basic mode, the method call activity is invoked. While loading the page in the advanced mode, the custom bounded task flow directly invokes the view activity. This ensures that the user entered data persists in the view objects across modes.

If the custom task flow UI does not render correctly, check whether transactional properties have been set in the custom task flow, such as the `requires-transaction` property, and so on.

Remove transactional properties from the task flow definition and set the data control scope to shared.

As the parent scheduled request submission UI task flow already has a transaction, Oracle ADF will commit all called task flow transactions if the data controls are shared.

---

**Note:**

When using the UI to schedule a job to run for a year, for example, a maximum of 300 occurrences display when clicking **Customize Times**.

---

65.22 About Securing Oracle ADF UIs

When creating Oracle ADF UIs for scheduled jobs, you can secure the individual task flows involved using a security policy.

The task flows you can secure are the following:
65.23 About Integrating Scheduled Job Logging with Oracle Fusion Applications

Oracle Enterprise Scheduler is fully integrated with Oracle Fusion Applications logging. The logger captures Oracle Enterprise Scheduler-specific attributes when invoking logging from within the context of a running job request. You can set the values to these Oracle Enterprise Scheduler attributes within the context of defining a job.

Jobs can generate a log file on the file system that can be viewed with the Monitoring UI.

In a typically configured Oracle Enterprise Scheduler hosting application, log and output files are stored in an Oracle WebCenter Content repository rather than on the file system. These files are available to end users through a page you provide for monitoring scheduled job requests. For more information about request monitoring, see About Monitoring Scheduled Job Requests Using an UI.

65.24 About Logging Scheduled Jobs and Writing to Output Files

Log messages written using the request log file APIs are written to the request log file and Oracle Fusion Applications logging at a severity level of FINE (only if logging is enabled at a level of FINE or lower).

For more information about managing log files, see the chapter "Managing Log Files and Diagnostic Data" in Administering Oracle Fusion Middleware.
65.24.1 About Using the Request Log

Note:

Do not use the request log for debugging and internal error reporting. For Oracle Enterprise Scheduler jobs, the request log is equivalent to the end-user UI for online applications. When writing Oracle Enterprise Scheduler job code, you should ideally log only translatable end user-oriented messages to the request log. You should not use the request log for debug messages or internal error messages that are oriented to system administrators and/or Oracle Support. The audience for debug messages and detailed internal error messages is typically system administrators and Oracle Support, not end users.

Therefore, debug and detailed internal error messages should be logged to the log called FND_LOG only.

For Oracle Enterprise Scheduler jobs, the request log is equivalent to the end user interface for web applications. When developing an Oracle Enterprise Scheduler job, log to the request log only translatable end-user oriented messages.

For example, if an end user enters a bad parameter to the Oracle Enterprise Scheduler job, a translated error message logged to the request log is displayed to the end user. The end user can then take the relevant corrective action.

Example 65-51 shows how to set log messages using the request log.

If the Oracle Enterprise Scheduler job fails due to an internal software error, log the detailed failure message to the log called FND_LOG for the system administrator or support. You can also log a high-level generic message to the request log so as to inform end users of the error. An example of a generic error message intended for end users: "Your request could not be completed due to an internal error."

Example 65-51 Setting Log Messages Using the Request Log

```
-- Seeded message to be displayed to the end user.
FND_MESSAGE.SET_NAME('FND', 'INVALID_PARAMETER');
-- Runtime parameter information
FND_MESSAGE.SET_TOKEN('PARAM_NAME', pName);
FND_MESSAGE.SET_TOKEN('PARAM_VALUE', pValue);
-- The following is useful for auto-logging errors.
FND_MESSAGE.SET_MODULE('fnd.plsql.mypackage.myfunctionA');
fnd_file.put_line( FND_FILE.LOG, FND_MESSAGE.GET );
```

65.24.2 About Using the Output File

Note:

Do not use the output file for debugging and internal error reporting.

The output file is a formally formatted file generated by an Oracle Enterprise Scheduler job. An output file can be sent to a printer or viewed in a UI window. Example 65-52 shows an invoice sent to an output file.
65.24.3 About Debugging and Error Logging

Debug and error logging should be done using the Oracle Diagnostic Logging APIs only. The Oracle Enterprise Scheduler request log should not be used for system administrator or Oracle support-oriented debug and error logging purposes. The request log is for the end users and it should only contain messages that are clear and concise. When an error occurs in an Oracle Enterprise Scheduler job, use an appropriate high-level (and, ideally, translated) message to report the error to the end user through the request log. The details of the error and any debug messages should be logged with Oracle Diagnostic Logging APIs.

Common PL/SQL, Java, or C code that could be invoked by both Oracle Enterprise Scheduler jobs and interactive application code should only use Oracle Diagnostic Logging APIs. If needed, the wrapper Oracle Enterprise Scheduler job should perform appropriate batching and logging to the request log for progress reporting purposes.

Using Logging in a Java Application

In Java jobs, use the log called AppsLog for debugging and error logging. You can retrieve an AppsLog instance from the CpContext object, by calling the method getLog().

Example 65-53 shows the use of logging in a Java application.

---

Note:

Example 65-53 uses an active WebAppsContext object. Do not attempt to log messages using an inactive or freed WebAppsContext object, as this can cause connection leaks.

---

Using Logging in a PL/SQL Application

PL/SQL APIs are part of the FND_LOG package. These APIs require invoking relevant application user session initialization APIs—such as the method FND_GLOBAL.INITIALIZE()—to set up user session properties in the database session.

These application user session properties, including UserId, RespId, AppId, SessionId, are needed for the log APIs. Typically, Applications Core invokes these session initialization APIs.

Log plain text messages with the method FND_LOG.STRING(). Log translatable message dictionary messages with the method FND_LOG.MESSAGE().

FND_LOG.MESSAGE() logs messages in encoded, but not translated, format, and allows the Log Viewer UI to handle translating messages based on the language preferences of the system administrator viewing the messages.

For details regarding the FND_LOG API, run the script $fnd/patch/115/sql/AFUTLOGB.pls at the prompt. Example 65-54 shows the PL/SQL logging syntax.

Example 65-55 shows how to log a message in PL/SQL after the AOL session has been initialized.

The global variable FND_LOG.G_CURRENT_RUNTIME_LEVEL allows callers to avoid a function call for messages at a lower level than the current configured level. If logging
is disabled, the current runtime level is set to a large number such as 9999 so that it is sufficient to simply log messages with levels greater than or equal to this number. This global variable is automatically populated by the FND_LOG_REPOSITORY package during session and context initialization.

**Example 65-56** shows sample code that illustrates the use of the global variable FND_LOG.G_CURRENT_RUNTIME_LEVEL.

---

**Note:**
For PL/SQL in a forms client, use the same APIs. Use the method FND_LOG.TEST() to check whether logging is enabled.

---

**Example 65-57** shows logging message dictionary messages.

**Using Logging in C**

**Example 65-58** illustrates the use of logging in a C application.

**Example 65-53  Logging in Java Using AppsLog**

define boolean authenticate(AppsContext ctx, String user, String passwd) throws SQLException, NoSuchUserException {
    AppsLog alog = (AppsLog) ctx.getLog();
    if(alog.isEnabled(Log.PROCEDURE))     /* To avoid String Concat if not enabled */
        alog.write("fnd.security.LoginManager.authenticate.begin",
                    "User=" + user, Log.PROCEDURE);
    /* Never log plain-text security sensitive parameters like passwd! */
    try {
        validUser = checkinDB(user, passwd);
    } catch(NoSuchUserException nsue) {
        if(alog.isEnabled(Log.EXCEPTION))
            alog.write("fnd.security.LoginManager.authenticate",nsue, Log.EXCEPTION);
    throw nsue; // Allow the caller to Handle it appropriately
    } catch(SQLException sqle) {
        if(alog.isEnabled(Log.UNEXPECTED)) {
            alog.write("fnd.security/LoginManager.authenticate", sqle, Log.UNEXPECTED);
            Message Msg = new Message("FND", "LOGIN_ERROR"); /* System Alert */
            Msg.setToken("ERRNO", sqle.getErrorCode(), false);
            Msg.setToken("REASON", sqle.getMessage(), false);
            /* Message Dictionary messages should be logged using write(..Message..),
             * and never using write(..String..) */
            alog.write("fnd.security/LoginManager.authenticate", Msg, Log.UNEXPECTED);
        }
    throw sqle; // Allow the caller to handle it appropriately
    } // End of catch(SQLException sqle)
    if(alog.isEnabled(Log.PROCEDURE))     /* To avoid String Concat if not enabled */
        alog.write("fnd.security/LoginManager.authenticate.end",
                    "validUser=" + validUser, Log.PROCEDURE);
    return success;
}

**Example 65-54  PL/SQL Logging Syntax**

```
PACKAGE FND_LOG IS
    LEVEL_UNEXPECTED CONSTANT NUMBER := 6;
    LEVEL_ERROR CONSTANT NUMBER := 5;
    LEVEL_EXCEPTION CONSTANT NUMBER := 4;
    LEVEL_EVENT CONSTANT NUMBER := 3;
```
LEVEL_PROCEDURE CONSTANT NUMBER := 2;
LEVEL_STATEMENT CONSTANT NUMBER := 1;

/*
** Writes the message to the log file for the specified
** level and module
** if logging is enabled for this level and module
*/
PROCEDURE STRING(LOG_LEVEL IN NUMBER,
                  MODULE    IN VARCHAR2,
                  MESSAGE   IN VARCHAR2);

/*
** Writes a message to the log file if this level and module
** are enabled.
** The message gets set previously with FND_MESSAGE.SET_NAME,
** SET_TOKEN, etc.
** The message is displayed from the message dictionary stack,
** if POP_MESSAGE is TRUE.
** Pass FALSE for POP_MESSAGE if the message will also be
** displayed to the user later.
** Example usage:
** FND_MESSAGE.SET_NAME(...); -- Set message
** FND_MESSAGE.SET_TOKEN(...); -- Set token in message
** FND_LOG.MESSAGE(..., FALSE); -- Log message
** FND_MESSAGE_RAISE_ERROR; -- Display message
*/
PROCEDURE MESSAGE(LOG_LEVEL   IN NUMBER,
                  MODULE      IN VARCHAR2,
                  POP_MESSAGE IN BOOLEAN DEFAULT NULL);

/*
** Tests whether logging is enabled for this level and module,
** to avoid the performance penalty of building long debug
** message strings unnecessarily.
*/
FUNCTION TEST(LOG_LEVEL IN NUMBER, MODULE IN VARCHAR2)
RETURN BOOLEAN;

Example 65-55  Logging a Message in PL/SQL After the AOL Session Has Been
Initialized

begin
/* Call a routine that logs messages. */
/* For performance purposes, check whether logging is enabled. */
if( FND_LOG.LEVEL_PROCEDURE >= FND_LOG.G_CURRENT_RUNTIME_LEVEL ) then
  FND_LOG.STRING(FND_LOG.LEVEL_PROCEDURE,
                 'fnd.plsql.MYSTUFF.FUNCTIONA.begin', 'Hello, world!');
end if;
/

Example 65-56  Logging a Message in PL/SQL Using
FND_LOG.G_CURRENT_RUNTIME_LEVEL

if( FND_LOG.LEVEL_STATEMENT >= FND_LOG.G_CURRENT_RUNTIME_LEVEL ) then
  dbg_msg := create_lengthy_debug_message(...);
  FND_LOG.STRING(FND_LOG.LEVEL_STATEMENT
                 'fnd.form.ABCDEFGH.PACKAGEA.FUNCTIONB.firstlabel', dbg_msg);
end if;
Example 65-57  Logging Message Dictionary Messages

```sql
if( FND_LOG.LEVEL_UNEXPECTED >=
    FND_LOG.G_CURRENT_RUNTIME_LEVEL) then
    FND_MESSAGE.SET_NAME('FND', 'LOGIN_ERROR'); -- Seeded Message
    -- Runtime Information
    FND_MESSAGE.SET_TOKEN('ERRNO', sqlcode);
    FND_MESSAGE.SET_TOKEN('REASON', sqlerrm);
    FND_LOG.MESSAGE(FND_LOG.LEVEL_UNEXPECTED,
        'fnd.plsql>Login.validate', TRUE);
end if;
```

Example 65-58  Logging in C

```c
#define  AFLOG_UNEXPECTED  6
#define  AFLOG_ERROR       5
#define  AFLOG_EXCEPTION   4
#define  AFLOG_EVENT       3
#define  AFLOG_PROCEDURE   2
#define  AFLOG_STATEMENT   1

/*
 ** Writes a message to the log file if this level and module is
 ** enabled
 */
void aflogstr(/*_ sb4 level, text *module, text* message _*/);

/*
 ** Writes a message to the log file if this level and module is
 ** enabled.
 ** If pop_message=TRUE, the message is popped off the message
 ** Dictionary stack where it was set with afdstring() afdtoken(),
 ** etc. The stack is not cleared (so messages below will still be
 ** there in any case).
 */
void aflogmsg(/*_ sb4 level, text *module, boolean pop_message _*/);

/*
 ** Tests whether logging is enabled for this level and module, to
 ** avoid the performance penalty of building long debug message
 ** strings
 */
boolean aflogtest(/*_ sb4 level, text *module _*/);

/*
 ** Internal
 ** This routine initializes the logging system from the profiles.
 ** It will also set up the current session and user name in its state */
void afloginit();
```
This chapter explains how Oracle Enterprise Scheduler security features provide access control for Oracle Enterprise Scheduler resources and application identity propagation for job execution.

- Introduction to Security
- Configuring Metadata Security for
- Configuring PL/SQL Job Security for
- Elevating Privileges for Jobs
- Configuring a Single Policy Stripe in
- Configuring Oracle Fusion Data Security for Job Requests

66.1 Introduction to Oracle Enterprise Scheduler Security

Oracle Enterprise Scheduler security includes the following:

- Protected operations on MetadataService; protected by MetadataPermission, which enforces access control on metadata objects. Only privileged users may create, delete, and update jobs and schedule metadata. For more information, see Oracle Enterprise Scheduler Metadata Access Control.

- Access control for job requests, enforced by Oracle Fusion Data Security policies. For more information about using Oracle Fusion Data Security policies, see Configuring Oracle Fusion Data Security for Job Requests.

- Support for the use of an application identity. Using an application identity enables elevated privileges to complete a job that requires higher privileges than those allotted to the user who submitted the job. For more information, see Job Execution Security.

66.1.1 Oracle Enterprise Scheduler Metadata Access Control

At design time, the metadata creator must decide which job functions can access which metadata objects. This is expressed by associating each metadata object with one or more roles and specifying one or more actions for each role. Figure 66-1 shows the metadata security summary.
The summary of design metadata security for Oracle Enterprise Scheduler is as follows:

1. Define Oracle Enterprise Scheduler metadata objects.
2. Create grants to control access to all metadata.
3. Define principals to be used for security.
4. Create grants to assign permissions for the metadata objects.
5. Assign relevant roles to the metadata, and specify actions for each role.
6. Assemble the EAR file for the application, including any security-related files.
7. Deploy the EAR file to Oracle WebLogic Server, which verifies grants in the Oracle Metadata Store and permissions assigned to roles with the LDAP server.

### 66.1.2 Oracle Enterprise Scheduler Job Execution Security

When submitting a job request, the submitting user is the user under whose permissions the job request is submitted. At job request execution time, all code, including pre-processing, post-processing, Java jobs, and substitution, runs as the submitting user, retaining all roles and credentials.

If the job metadata specifies `SYS_RUNAS_APPLICATIONID`, however, the job runs under the elevated privileges of an application ID. For more information, see [Elevating Privileges for Jobs](#).

### 66.2 Configuring Metadata Security for Oracle Enterprise Scheduler

When a user accesses Oracle Enterprise Scheduler services using the `RuntimeService` or `MetadataService`, the identity of the user is acquired and
Oracle Enterprise Scheduler checks whether the user has the required permissions to access resources such as metadata objects. For example, if a user named teller1 must call `getJobDefinition` to access a metadata object named `calculateFees`, Oracle Enterprise Scheduler ensures that teller1 has READ permission for the metadata object `calculateFees` before returning the object.

At design time, you must determine the job functions for which you want to enable access to particular metadata objects by associating each metadata object with one or more roles and specifying one or more actions for each role.

There are two options for metadata role assignments:

- Using Oracle JDeveloper Tools Oracle ADF Security Wizard
- Using Oracle JDeveloper Oracle Enterprise Scheduler add-in metadata pages

Oracle JDeveloper ADF Security wizard creates the roles you use; the roles must be created before you can register roles with a metadata object.

### 66.2.1 How to Enable Application Security with Oracle ADF Security Wizard

These steps describe a minimal, validated security setup for an application using Oracle Enterprise Scheduler.

Follow these steps to create a working `jps-config.xml` and a partially-populated `jazn-data.xml`. Use these steps to configure servlets to work with JPS.

1. In Oracle JDeveloper, with an application open, from the Application menu select Secure.
2. From the dropdown list, select Configure ADF Security. The Configure ADF Security wizard displays.
3. In the Enable ADF Security page, select either ADF Authentication and Authorization or ADF Authentication and click Next.
4. In the Select authentication type page, select either HTTP Basic Authentication or Form-Based Authentication and click Next.
5. In the Enable automatic policy grants page, select the appropriate options from the Enable Automatic Grant area, and click Next.
6. In the Specify authenticated welcome page, select options as needed and click Next.
7. In the Summary page verify the options and click Finish.
8. In the Security Infrastructure Created dialog, click OK.

Next, to enable security and ensure that the `jazn-data.xml` is included in the application deployment, perform the following steps after assembling the EAR file for the application.

**Ensure the security-related files are included with the EAR file:**

1. In Oracle JDeveloper, select Application > Application Properties.
2. In the Application Properties page, in the Navigator select Deployment.
3. In the Deployment Profiles area, select the EAR file Deployment descriptor.
4. Click Edit.
   The Edit EAR Deployment Profile Properties page displays.

5. In the Edit EAR Deployment Profile Properties page, expand File Groups > Application Descriptors > Filters.

6. In the Filters area, select the Files tab.

7. Ensure that the files jazn-data.xml, jps-config.xml, and weblogic-application.xml are selected under the META-INF folder.

8. Click OK to save the descriptor.

### 66.2.2 How to Define Principals for Security

You need to define roles before the roles are used in Oracle Enterprise Scheduler security. You can define two types of roles:

- Enterprise roles: These are defined directly in Oracle WebLogic Server either using the Oracle WebLogic Server console, using the WLST scripts, or using the Oracle ADF Security Wizard in Oracle JDeveloper.
- Application roles: These can be defined in the jazn-data.xml file or using the Oracle ADF Security Wizard.

**To define principals security:**

1. In Oracle JDeveloper, open the application and expand Application Resources in the Application Navigator.

2. In the Application Resources area, expand Descriptors and META-INF.

3. In META-INF, double-click to open jazn-data.xml.

4. In the page showing jazn-data.xml, select the Overview tab. If the Overview tab is not shown, try closing jazn-data.xml and then opening it again.

5. Click Application Roles...(Manage Users and Roles).


7. In the navigator, select Roles and click Add....
   The Add Role window displays.

8. In the Add Role window, enter a name in the Name field.

9. Click OK.

10. On the Edit JPS Identity and Policy Store page, in the navigator select Application Policy Store. If there is a sub-element with the same name as the application, go to the next step, Otherwise, do the following:
   a. Select Application Policy Store.
   b. Click New.
      The Create Application Policy window displays.
c. In the Create Application Dialog the **Display Name** field should contain the application name.

d. Click **OK** to accept the default Display Name.

11. On the Edit JPS Identity and Policy Store page, in the navigator expand **Application Policy Store** and expand the **application name**.

12. In the navigator, select **Application Roles**.

   The Application Roles page displays.

13. In the Application Roles page, click **Add**... to add roles. For correct functionality, at least one enterprise role must be mapped to the application role by adding enterprise roles in the **Member Roles** tab.

14. Click **OK**.

### 66.2.3 How to Create Grants with Oracle Enterprise Scheduler Metadata Pages

Access to all metadata is controlled by grants. To ensure access by the right identities, you need to give the correct grants.

First, create any required Oracle Enterprise Scheduler metadata in an application using **File > New > Business Tier > Enterprise Scheduler Metadata**. For more information about defining metadata, see About Creating a Job Definition.

Using Oracle JDeveloper, you can add security grants to Oracle Enterprise Scheduler metadata objects.

**To secure Oracle Enterprise Scheduler metadata objects:**

1. Open the Editor page for any Oracle Enterprise Scheduler metadata object.

2. In the Access Control area, click **Add** to add a new access control item.

3. In the Add Access Control dialog, select a Role from the dropdown list. This selects a role to grant access privileges.

4. Select one or more actions from the list, **Read**, **Execute**, **Update**, or **Delete**.

5. Click **OK**. This displays the updated role, as shown in **Figure 66-2**.

6. Repeat for as many roles as needed.
There may be occasions when you want to create grants explicitly, for example when using wild cards. These steps show how to set up grants using the Oracle ADF Security wizard.

Note that these steps assume you have already created application roles.

To specify grants with the Oracle ADF Security wizard:

1. In the Application Navigator, expand the Application Resources panel.
2. Expand Descriptors and META-INF, as shown in Figure 66-3.
3. Double-click `jazn-data.xml` to open the file. In the editor panel for `jazn-data.xml`, select the Overview tab, and click Application Roles... (Manage Users and Roles). This displays the JPS Identity & Policy Store dialog. Note, if the Overview tab is not shown, try closing `jazn-data.xml` and then opening it again.

4. In the JPS Identity & Policy Store dialog, in the navigator expand Application Policy Store.

5. Expand `application-name`, and select Application Roles.

6. Click New.

7. Enter the display name you wish for this grant, and click OK.

8. Select the Principals tab, and click Add.

9. Enter the name of the application role which will receive the grant; this should be one of the role names created. Leave the Class field as is.

10. Click OK.

11. With the new role selected in the Principals tab, make sure the Type is role.

12. Select the Permissions tab, and click Add....

13. For the Name field, enter a full permission string or a partial string with wildcards; see Table 66-1 for examples. In the Class field, enter `oracle.as.scheduler.security.MetadataPermission`. Click OK.

14. With the new permission selected in the Permissions tab, enter the desired actions in the Actions Field.

15. Click OK to save the grant.
**Note:**
If necessary, use the following workaround:

1. Right-click the jazn-data.xml file and select **Open**.
2. Click the Source tab.
3. Under `<jazn-policy><grant><grantee>`, remove the elements `<display-name>` and `<type>`.

### Table 66-1 Sample Permission Grants for Security Using Oracle ADF

<table>
<thead>
<tr>
<th>Name</th>
<th>Actions</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>packag-&lt;br/&gt;part.JobDefinition.MyJavaSucJobDef</td>
<td>EXECUTE</td>
<td>Grants the ability to submit requests for a single metadata item.</td>
</tr>
<tr>
<td>mypackage.subpackage.*</td>
<td>CREATE, EXECUTE</td>
<td>Grants the ability to create and execute any new metadata items in /mypackage/subpackage.</td>
</tr>
<tr>
<td>JobDefinition.SYS_AdHo&lt;br/&gt;cRequest</td>
<td>CREATE, EXECUTE</td>
<td>Grants ad hoc submission permission</td>
</tr>
<tr>
<td>mypackage.*</td>
<td>CREATE, EXECUTE, DELETE</td>
<td>Grants wide-open permissions</td>
</tr>
</tbody>
</table>

### 66.2.5 MetadataPermission APIs

Grants for metadata are part of the class `oracle.as.scheduler.security.MetadataPermission`. The name, or target of the permission is based on the package, metadata object type and name of the metadata object being protected. This identifier can be retrieved from `MetadataObjectId#toPermissionString()`.

**Table 66-2** lists the actions for the grants. The notation `<Type>` is a placeholder for all of the metadata object types. For example, `get<Type>()` refers to the methods `getJobDefinition()`, `getJobType()`, `getJobSet()`.

<table>
<thead>
<tr>
<th>Action</th>
<th>Implies</th>
<th>Metadata Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ</td>
<td>None</td>
<td><code>get&lt;Type&gt;()</code>, <code>query&lt;Type&gt;()</code></td>
</tr>
<tr>
<td>EXECUTE</td>
<td>READ</td>
<td><code>submitRequest()</code></td>
</tr>
<tr>
<td>CREATE</td>
<td>READ</td>
<td><code>add&lt;Type&gt;()</code></td>
</tr>
<tr>
<td>UPDATE</td>
<td>READ</td>
<td><code>update&lt;Type&gt;()</code></td>
</tr>
<tr>
<td>DELETE</td>
<td>READ</td>
<td><code>delete&lt;Type&gt;()</code></td>
</tr>
</tbody>
</table>

If you are submitting ad-hoc requests, you can have full wildcard ("*") permission with both `EXECUTE` and `CREATE` actions. When submitting ad-hoc requests, that is, using
submitRequest() without certain MetadataObjectIds, you can grant permissions with the full wildcard ("*") name using the EXECUTE and CREATE actions.

66.2.6 What Happens When You Configure Metadata Security

Each time a user application calls a MetadataService or RuntimeService method, Oracle Enterprise Scheduler checks the current subject for privileges on the metadata accessed by the methods. For example, submitting a request requires EXECUTE permissions on the job definition or job set metadata object associated with the submission. Methods that change metadata, for example calling updateJobDefinition(), require UPDATE permissions.

For all MetadataService methods except queries, an exception is thrown when the user tries to access a metadata object for which the user does not have permission.

The MetadataService query methods have different behavior. When a user performs a query Oracle Enterprise Scheduler only returns metadata objects that have READ permission. Thus a user who has no permissions on metadata objects receives an empty list for all queries, but this user would not see an exception thrown due to lack of permissions.

The value of SystemProperty.USER_NAME is overwritten at submission time; the user cannot spoof an identity at submission time using SystemProperty.USER_NAME.

66.2.7 Related Links

The following document provides additional information related to subjects discussed in this section:

For more information, see "Assembling Oracle Enterprise Scheduler Oracle Fusion Applications" in the Extensibility Guide for Developers.

66.3 Configure Web Service Security for Oracle Enterprise Scheduler

For information about securing the Oracle Enterprise Scheduler web service, see the chapter "Securing the Oracle Enterprise Scheduler Web Service" in Developing Applications for Oracle Enterprise Scheduler.

66.4 Configure PL/SQL Job Security for Oracle Enterprise Scheduler

For standalone cases, implement the application user session using Java or the PL/SQL API as described in Implementing Application User Sessions.

66.5 Elevating Privileges for Oracle Enterprise Scheduler Jobs

When a user accesses Oracle Enterprise Scheduler services using the RuntimeService or MetadataService interfaces, the identity of the user calling the methods is acquired. This identity is used to check whether the user has the required permissions to access certain resources such as metadata objects. For example, if user teller1 calls the method getJobDefinition for metadata object caculateFees, Oracle Enterprise Scheduler ensures that teller1 has read permissions for metadata object caculateFees before returning the object.

The caller identity is also used to run jobs requested by the user. For example, if user teller1 calls the method submitRequest() for a Java job, the requested jobs run under teller1 and retain all roles and credentials assigned to that user.
Oracle Enterprise Scheduler supports the use of an application identity. Using an application identity enables elevated privileges for completion of a job that requires higher privileges than those allotted to the submitting user.

For more information about enabling elevating privileges, see About Elevating Access Privileges for a Scheduled Job.

66.6 Configuring a Single Policy Stripe in Oracle Enterprise Scheduler

Oracle Platform Security policy store serves as the repository for authorization policies. Authorization policies load at run time into the Java Virtual Machine, and are used to make decisions regarding authorization. Authorization policies comprise a hierarchy of application roles, the mapping of enterprise roles to application roles and permissions grants to application roles. Application roles can also be hierarchical.

Aside from authorization policies, Oracle Platform Security policy store also stores administrative constructs that help in maintaining these authorization policies, including resource catalogs (with associated resource types), permission sets and role categories. The authorization polices and administrative components are scoped to an application. This is known as an application stripe.

An application stripe is a collection of JAAS policies applicable to the application with which it is associated. Out of the box, an application stripe maps to an Oracle Java EE application. Oracle Platform Security also supports mapping multiple Java EE applications to one application stripe. The application ID string identifies the name of the application or applications.

66.6.1 How to Configure a Single Policy Stripe in Oracle Enterprise Scheduler

Oracle Enterprise Scheduler allows specifying an applicationStripe name and mapping it to a JPS policy context ID. You can assign multiple Oracle Enterprise Scheduler hosting applications to a single policy context.

To configure an Oracle Enterprise Scheduler hosting application to a specific applicationStripe:

1. Open the ejb-jar.xml file.
2. Under the message-driven element, add an activation-config-properties element with the value applicationStripe.
3. Under the jpsinterceptor-class element, configure the JpsInterceptor.
   Make sure to match the value of applicationStripe under the <message-driven> element with the application.name value under the <interceptor> element.
   Example 66-1 shows an applicationStripe configuration for the policy context ESS_FUNCTIONAL_TEST_APP_STRIPE.
4. If your application has a web module, configure the web module JpsFilter to use the same applicationStripe in the file web.xml. Example 66-2 shows a code sample.

Example 66-1 Configuring the applicationStripe and the JpsInterceptor

```
<ejb-jar>
  ....
```
<enterprise-beans>
<message-driven>
  <ejb-name>ESSAppEndpoint</ejb-name>
  <ejb-class>oracle.as.scheduler.ejb.EssAppEndpointBean</ejb-class>
  <activation-config>
    ....
    <activation-config-property>
      <activation-config-property-name>applicationStripe</activation-config-property-name>
      <activation-config-property-value>ESS_FUNCTIONAL_TESTS_APP_STRIPE</activation-config-property-value>
    </activation-config-property>
  </activation-config>
</message-driven>
....
</enterprise-beans>

<interceptors>
  <interceptor>
    <interceptor-class>oracle.security.jps.ee.ejb.JpsInterceptor</interceptor-class>
    <env-entry>
      <env-entry-name>application.name</env-entry-name>
      <env-entry-type>java.lang.String</env-entry-type>
      <env-entry-value>ESS_FUNCTIONAL_TESTS_APP_STRIPE</env-entry-value>
    </env-entry>
  </interceptor>
</interceptors>
</ejb-jar>

Example 66-2 Configuring the Web Module in web.xml

<web-app>
  <filter>
    <filter-name>JpsFilter</filter-name>
    ...
    <init-param>
      <param-name>application.name</param-name>
      <param-value>ESS_FUNCTIONAL_TESTS_APP_STRIPE</param-value>
    </init-param>
  </filter>
</web-app>

66.6.2 What Happens When You Configure a Single Policy Stripe

At design time, an application stripe manifests as:

- An `<application>` element under the `<policystore>` element in the jazndata.xml file.
- A node under the node `cn=<Weblogic.domain.name>,cn=JPSContext,cn=<root.node>, such as cn=ATGDemo, cn=base_domain, cn=JPSContext, cn=MY_Node.`
66.6.3 What Happens at Runtime

At run time, an application stripe manifests as an instance of the class `oracle.security.jps.service.policystore.ApplicationPolicy`.

66.7 Configure Oracle Fusion Data Security for Job Requests

Oracle Fusion Data Security for Oracle Fusion application enforces security authorizations for access and modification of specific data records. Oracle Fusion Data Security integrates with Oracle Platform Security Services (OPSS) by granting actions to OPSS principals. The grant defines who (the principals) can do what (the actions) on a given resource. A grant in Oracle Fusion Data Security can use any enterprise user or enterprise group as principals. For more information about implementing Oracle Fusion Data Security, see Implementing Oracle Fusion Data Security.

In the context of Oracle Enterprise Scheduler, a job request access control data security policy comprises a grant, a grantee and a set of `ESS_REQUEST` privileges for a set of job requests as follows:

- A grantee, represented by grantee ID such as a user or application role, the ID should match the user GUID or application role GUID retrieved from Oracle Fusion Middleware.

- A set of `ESS_REQUEST` privileges represented by a menu ID mapped to a set of form functions.

- A set of data represented by an `INSTANCE_SET` ID. An `INSTANCE_SET` is typically represented by a predicate which can be appended to a query to the job request data exposed to Oracle Fusion Applications (see Oracle Fusion Data Security Artifacts).

The job request access control data security policy can be managed using Oracle Authorization Policy Manager as are other Oracle Fusion Data Security policies. If Oracle Authorization Policy Manager is not available, you can use SQL scripts to manipulate the Oracle Fusion Data Security artifacts.

66.7.1 Oracle Fusion Data Security Artifacts

To use Oracle Enterprise Scheduler job request access control feature in the context of Oracle Fusion Applications, the Oracle Fusion Applications schema and Oracle Enterprise Scheduler schema must be located in a single database.

Oracle Enterprise Scheduler implements job request data security on top of the `request_history` and `request_property` tables. It exposes Oracle Enterprise Scheduler job request related data to the Oracle Fusion Applications schema through the following views: `request_history_view` and `request_property_view`. Two synonyms are created in the Oracle Fusion Applications schema which are linked to the Oracle Enterprise Scheduler schema.

The `request_history_view` contains all columns that correspond to `RuntimeService.QueryField`, which is used when constructing the filter for `queryRequest()` operations, as well as two other columns: `submitter` and `submitterguid`. Be sure to define your `INSTANCE_SET` based on these columns only.
Table 66-3 lists the Oracle Fusion Applications schema tables and their Oracle Enterprise Scheduler synonyms, as well as the columns used to define data security policies.

### Table 66-3  Mapping Oracle Fusion Applications Schema Synonyms to Oracle Enterprise Scheduler Schema Views and Relevant Columns

<table>
<thead>
<tr>
<th>Oracle Fusion Applications Schema Synonym</th>
<th>Link to Oracle Enterprise Scheduler Schema View</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>ess_request_history</td>
<td>request_history_view</td>
<td>See table for the QueryField and View Column mapping.</td>
</tr>
<tr>
<td>ess_request_property</td>
<td>request_property_view</td>
<td>create or replace view request_property_view as select requestid, name, scope, datatype, value, lobvalue, lobflag from request_property with read only;</td>
</tr>
</tbody>
</table>

Table 66-4 shows the mapping of RuntimeService.QueryField columns to the Oracle Enterprise Scheduler request_history_view columns.

### Table 66-4  Mapping RuntimeService.QueryField Columns to request_history_view Columns

<table>
<thead>
<tr>
<th>RuntimeService.QueryField Columns</th>
<th>Request_history_view Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryField.REQUESTID</td>
<td>requestid</td>
</tr>
<tr>
<td>QueryField.APPLICATION</td>
<td>application</td>
</tr>
<tr>
<td>QueryField.USERNAME</td>
<td>userName</td>
</tr>
<tr>
<td>QueryField.PRODUCT</td>
<td>product</td>
</tr>
<tr>
<td>QueryField.REQUESTCATEGORY</td>
<td>requestCategory</td>
</tr>
<tr>
<td>QueryField.PRIORITY</td>
<td>priority</td>
</tr>
<tr>
<td>QueryField.NAME</td>
<td>name</td>
</tr>
<tr>
<td>QueryField.ABSPARENTID</td>
<td>absParentId</td>
</tr>
<tr>
<td>QueryField.TYPE</td>
<td>type</td>
</tr>
<tr>
<td>QueryField.DEFINITION</td>
<td>definition</td>
</tr>
<tr>
<td>QueryField.STATE</td>
<td>state</td>
</tr>
</tbody>
</table>
Table 66-4  (Cont.) Mapping RuntimeService.QueryField Columns to request_history_view Columns

<table>
<thead>
<tr>
<th>RuntimeService.QueryField Columns</th>
<th>request_history_view Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>QueryField.SCHEDULE</td>
<td>schedule</td>
</tr>
<tr>
<td>QueryField.PROCESSESTART</td>
<td>processStart</td>
</tr>
<tr>
<td>QueryField.PROCESSSEND</td>
<td>processEnd</td>
</tr>
<tr>
<td>QueryField.REQUESTEDSTART</td>
<td>requestedStart</td>
</tr>
<tr>
<td>QueryField.REQUESTEDEND</td>
<td>requestedEnd</td>
</tr>
<tr>
<td>QueryField.SUBMISSION</td>
<td>submission</td>
</tr>
<tr>
<td>QueryField.PARENTREQUESTID</td>
<td>parentRequestId</td>
</tr>
<tr>
<td>QueryField.WORKASSIGNMENT</td>
<td>workAssignment</td>
</tr>
<tr>
<td>QueryField.SCHEDULE</td>
<td>scheduled</td>
</tr>
<tr>
<td>QueryField.REQUESTTRIGGER</td>
<td>requesttrigger</td>
</tr>
<tr>
<td>QueryField.PROCESSOR</td>
<td>processor</td>
</tr>
<tr>
<td>QueryField.CLASSNAME</td>
<td>classname</td>
</tr>
<tr>
<td>QueryField.ELAPSEDETIME</td>
<td>elapsedtime</td>
</tr>
<tr>
<td>QueryField.WAITTIME</td>
<td>waittime</td>
</tr>
<tr>
<td>QueryField.SUBMITTER</td>
<td>submitter</td>
</tr>
<tr>
<td>QueryField.SUBMITTERGUID</td>
<td>submitterguid</td>
</tr>
</tbody>
</table>

Table 66-5 maps FND_MENUS to FND_FORM_FUNCTIONS as reflected in FND_MENU_ENTRIES.

Table 66-5  Mapping FND_MENUS to FND_FORM_FUNCTIONS

<table>
<thead>
<tr>
<th>FND_MENUS in the Oracle Fusion Applications Schema</th>
<th>FND_FORM_FUNCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_REQUEST_ADMIN</td>
<td>ESS_REQUEST_READ</td>
</tr>
<tr>
<td></td>
<td>ESS_REQUEST_UPDATE</td>
</tr>
<tr>
<td></td>
<td>ESS_REQUEST_HOLD</td>
</tr>
<tr>
<td></td>
<td>ESS_REQUEST_CANCEL</td>
</tr>
<tr>
<td></td>
<td>ESS_REQUEST_LOCK</td>
</tr>
<tr>
<td></td>
<td>ESS_REQUEST_RELEASE</td>
</tr>
<tr>
<td></td>
<td>ESS_REQUEST_DELETE</td>
</tr>
<tr>
<td></td>
<td>ESS_REQUEST_PURGE</td>
</tr>
<tr>
<td>ESS_REQUEST_VIEW</td>
<td>ESS_REQUEST_READ</td>
</tr>
</tbody>
</table>
### Table 66-5  (Cont.) Mapping FND_MENUS to FND_FORM_FUNCTIONS

<table>
<thead>
<tr>
<th>FND_MENUS in the Oracle Fusion Applications Schema</th>
<th>FND_FORM_FUNCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_REQUEST_OPERATE</td>
<td>ESS_REQUEST_READ</td>
</tr>
<tr>
<td></td>
<td>ESS_REQUEST_HOLD</td>
</tr>
<tr>
<td></td>
<td>ESS_REQUEST_CANCEL</td>
</tr>
<tr>
<td></td>
<td>ESS_REQUEST_LOCK</td>
</tr>
<tr>
<td></td>
<td>ESS_REQUEST_RELEASE</td>
</tr>
<tr>
<td>ESS_REQUEST_OUTPUT_ADMIN</td>
<td>ESS_REQUEST_OUTPUT_VIEW</td>
</tr>
<tr>
<td></td>
<td>ESS_REQUEST_OUTPUT_DELETE</td>
</tr>
</tbody>
</table>

Table 66-6 lists the required data privilege (form_function) for a user to perform an Oracle Enterprise Scheduler runtimeService operation.

### Table 66-6  Data Privileges Needed to Execute runtimeService Operations

<table>
<thead>
<tr>
<th>RuntimeService API Operation</th>
<th>Data Privilege (FND_FORM_FUNCTIONS)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>open</td>
<td>none</td>
<td></td>
</tr>
<tr>
<td>close</td>
<td>none</td>
<td>Two overloaded methods.</td>
</tr>
<tr>
<td>submitRequest</td>
<td>none</td>
<td>Five overloaded methods, which are secured by metadata security, not data security.</td>
</tr>
<tr>
<td>getRequestParam</td>
<td>ESS_REQUEST_READ</td>
<td></td>
</tr>
<tr>
<td>getRequestParamState</td>
<td>ESS_REQUEST_READ</td>
<td></td>
</tr>
<tr>
<td>getRequests</td>
<td>ESS_REQUEST_READ</td>
<td></td>
</tr>
<tr>
<td>getRequestMethod</td>
<td>ESS_REQUEST_READ</td>
<td></td>
</tr>
<tr>
<td>getRequestDetail</td>
<td>ESS_REQUEST_READ</td>
<td></td>
</tr>
<tr>
<td>lockRequest</td>
<td>ESS_REQUEST_LOCK</td>
<td></td>
</tr>
<tr>
<td>updateRequestParam</td>
<td>ESS_REQUEST_UPDATE</td>
<td></td>
</tr>
<tr>
<td>queryRequests</td>
<td>ESS_REQUEST_READ</td>
<td></td>
</tr>
<tr>
<td>holdRequest</td>
<td>ESS_REQUEST_HOLD</td>
<td></td>
</tr>
<tr>
<td>releaseRequest</td>
<td>ESS_REQUEST_RELEASE</td>
<td></td>
</tr>
<tr>
<td>cancelRequest</td>
<td>ESS_REQUEST_CANCEL</td>
<td></td>
</tr>
<tr>
<td>deleteRequest</td>
<td>ESS_REQUEST_DELETE</td>
<td></td>
</tr>
<tr>
<td>purgeRequest</td>
<td>ESS_REQUEST_PURGE</td>
<td></td>
</tr>
<tr>
<td>publishEvent</td>
<td>none</td>
<td>Not targeted to a request.</td>
</tr>
<tr>
<td>isHandleRollbackOnly</td>
<td>none</td>
<td>Not targeted to a request.</td>
</tr>
</tbody>
</table>
Table 66-6  (Cont.) Data Privileges Needed to Execute runtimeService Operations

<table>
<thead>
<tr>
<th>RuntimeService API Operation</th>
<th>Data Privilege (FND_FORM_FUNCTIONS)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>setHandleRollbackOnly</td>
<td>none</td>
<td>Not targeted to a request.</td>
</tr>
<tr>
<td>replaceSchedule</td>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>

Table 66-7 displays the INSTANCE_SET conditions provided by Oracle Authorization Policy Manager.

Table 66-7  INSTANCE_SET Conditions Provided by Oracle Authorization Policy Manager

<table>
<thead>
<tr>
<th>INSTANCE_SET Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQS_SUBMITTEDBY_SESSIONUSER</td>
<td>Oracle Enterprise Scheduler requests that the submitter is the current session user.</td>
</tr>
<tr>
<td>REQS_RUNAS_SESSIONUSER</td>
<td>Oracle Enterprise Scheduler requests that the RunAs user is the current session user.</td>
</tr>
<tr>
<td>REQS_SUBREQS_BY_SUBMITTER</td>
<td>Oracle Enterprise Scheduler requests and subrequests are all submitted by the submitter.</td>
</tr>
<tr>
<td>REQS_ALL_OF_ONE_APP</td>
<td>Indicates all Oracle Enterprise Scheduler requests related to a product within a logical application. This condition takes two parameters that match the job request parameter values of SYS_application and SYS_product.</td>
</tr>
<tr>
<td>ESS_REQS_BY_NAME_VALUE_PARAM</td>
<td>Oracle Enterprise Scheduler job request whose RequestParameter name value pair is specified in data security grants. This condition takes two parameters that match the one job request parameter's name and value.</td>
</tr>
</tbody>
</table>

Table 66-8 lists the Oracle Fusion Data Security policies available for use with Oracle Enterprise Scheduler out of the box.

Table 66-8  Oracle Fusion Data Security Policies for Oracle Enterprise Scheduler

<table>
<thead>
<tr>
<th>Oracle Fusion Data Security Policy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESS_REQUEST_SUBMITTER_ADMIN_SUBMITTED_REQUESTS</td>
<td>The submitter of the job request is permitted to view and administer the requests they submitted.</td>
</tr>
<tr>
<td>ESS_REQUEST_SUBMITTER_ADMIN_SUBMITTED_REQUESTS_SUBREQS</td>
<td>The submitter of the job requests and subrequests is permitted to view and administer on the requests they submitted.</td>
</tr>
<tr>
<td>ESS_REQUEST_RUNASUSER_ADMIN_EXECUTED_REQUESTS</td>
<td>The runAs user is permitted to view and administer the requests they execute.</td>
</tr>
<tr>
<td>ESS_REQUEST_RUNASUSER_VIEWOUTPUT_EXECUTED_REQUESTS</td>
<td>The runAs user is permitted to view the output of the job requests they executed.</td>
</tr>
</tbody>
</table>

For more information about the runAs user, or elevating access privileges, see About Elevating Access Privileges for a Scheduled Job.
66.7.2 How to Apply Oracle Fusion Data Security Policies

The Oracle Fusion Data Security components described in Oracle Fusion Data Security Artifacts can be applied as follows.

To apply Oracle Fusion Data Security policies:

1. Examine the policies described in Table 66-8 and determine whether you can use any of them in your application.
   - If you can use one of these policies, skip to the last step.
   - If the policies do not apply, continue to the next step.

2. Determine whether any of the FND_MENUS listed in Table 66-5 suit the out-of-the-box Oracle Fusion Data security policy you selected for your application. If you cannot apply any of the FND_MENUS listed in Table 66-5, create your own FND_MENUS and FND_MENUS_ENTRIES as described in Implementing Oracle Fusion Data Security.

3. Determine whether you can use the INSTANCE_SET conditions in Table 66-7 and the Oracle Fusion Data Security policies in your application. If you cannot use the conditions, create your own FND_INSTANCE_SET. For more information about creating an FND_INSTANCE_SET, see Implementing Oracle Fusion Data Security.

4. Create an Oracle Fusion Data Security policy, as described in How to Create Functional and Data Security Policies for Components.

Note:
If developing an Oracle Fusion application, do not grant an Oracle Enterprise Scheduler access policy to the grantee of an authenticated-role or anonymous-role, as doing so may affect the behavior of Oracle Enterprise Scheduler or other products.

5. Test your application.

66.7.3 How to Create Functional and Data Security Policies for Oracle Enterprise Scheduler Components

You can use Oracle Authorization Policy Manager to create functional and data security policies for Oracle Enterprise Scheduler.

To create functional and data security policies for Oracle Enterprise Scheduler:

1. Create a resource.
   a. From the list of policies, expand the fcsm policy stripe and select fcsm > Resource Catalog > Resources.
   b. From the Actions menu, click New.
   c. Define a resource with the resource type ESSMetadataResourceType, as well as the name and display name of the Oracle Enterprise Scheduler component using the following syntax:
d. Save the resource.

2. Define a resource policy.
   a. Select the resource you just created and click Create Policy.
   b. Add principals (grantees) by clicking the Add button.
   c. In the Add Principal window, search for the relevant application role or roles. Select the roles and click Add.
   d. In the Actions field, select the relevant actions and click Apply.

3. Create an authorization condition.
   a. In the Authorization Management tab, select Global and search for the database resource you want to use. Table 66-8 lists the database resources related to Oracle Enterprise Scheduler.
   b. Select the resource and click Edit.
   c. Click the Conditions tab and select Actions > New.
   d. Enter a name, display name and SQL predicate for the condition.

4. Define a data policy.
   a. From the Actions menu, select New Policy.
   b. In the New Policy window, use the Role and Database Resource fields to add the relevant roles and resources.
   c. Select the role you defined. In Database Resource Details region, select the condition name you just created and choose the actions you require.

66.7.4 Related Links

The following document provides additional information related to subjects discussed in this section:

For more information about creating policies in Oracle Authorization Policy Manager, see the section "Managing Authorization Policies" in the "Securing Oracle Fusion Applications" chapter of the Administrator’s Guide.
This chapter documents how developers can implement Oracle Social Network in an application by creating the Model project in JDeveloper and by displaying it in the Global Area of the UIShell.

The chapter includes these sections:

- Prerequisites for Oracle Social Network Integration
- Implementing the Oracle Social Network Model Project
- Implementing the Oracle Social Network Setup UI
- Adding Oracle Social Network to Your Application UI
- What Happens When Functional Administrators Manage Oracle Social Network Objects
- What Happens When End Users Interact with Oracle Social Network Object Instances
- Testing End-To-End Flow

67.1 Prerequisites for Oracle Social Network Integration

For Oracle Social Network integration to work, there first must be a connection defined between Oracle Fusion applications and Oracle Social Network. This connection normally is set up by the provisioner (see Table 67-1) and must be able to be configured because Oracle Fusion applications and Oracle Social Network will not reside on the same server. These configuration items must be in place:

- Oracle Social Network URL must be defined in Topology Manager with a Module Short Name of 'OSN'.
  
  On a production instance, there will be an internal and external Topology Manager URL for Oracle Social Network. The internal URL will be used for all data sent to Oracle Social Network. The external URL will be used when navigating from within Fusion Applications.

- The Oracle Social Network credential must exist in the WebLogic Server credential store.

- In the weblogic-application.xml file that is present at the workspace level, add a reference to the Oracle Social Network model shared library:

  <library-ref>
  <library-name>oracle.social.client</library-name>
  </library-ref>
In the `weblogic.xml` file for the UI project (that is mapped to the WAR deployment), add a reference to the Oracle Social Network view shared library. This entry is needed in any UI project that needs to test or run the Oracle Social Network integration using the UIShell as described in Adding Oracle Social Network to Your Application UI.

```xml
<library-ref>
<library-name>oracle.social.client.view</library-name>
</library-ref>
```

### 67.2 Implementing the Oracle Social Network Model Project

Each of the roles listed in Table 67-1 involves a different part of the entire Oracle Fusion Applications-Oracle Social Network integration work flow.

**Table 67-1  Oracle Social Network Integration Roles**

<table>
<thead>
<tr>
<th>Role</th>
<th>Description</th>
<th>Primary responsibilities for integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fusion Applications</td>
<td>A developer in a product team that has view object data that is to be sent to Oracle Social Network.</td>
<td>• Defining which view objects are to be available for Oracle Social Network integration.</td>
</tr>
<tr>
<td>Developer</td>
<td></td>
<td>• Configuring the view object and its Attributes for Oracle Social Network integration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Updating the product User Interface to include the Oracle Social Network <strong>Share</strong> button to allow for Manual Oracle Social Network integration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Configuring their Oracle Fusion Functional Setup Manager to have a task flow entry for &quot;Manage Social Network Objects&quot;.</td>
</tr>
<tr>
<td>Provisioner</td>
<td>The person who provides and configures the Oracle WebLogic Server environment in which the Oracle Fusion application is to be run.</td>
<td>• Creating the Oracle Social Network credential in the credential store (if not pre-provisioned).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Configuring Topology Manager to have a deployed application entry for &quot;Oracle Social Network.&quot;</td>
</tr>
<tr>
<td>Functional Administrator</td>
<td>The person on the site who will configure which view objects will be used at product runtime from the list of available view objects for Oracle Social Network integration.</td>
<td>• Choosing the type of integration each of the available view objects will have.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Selecting which attributes will be sent to Oracle Social Network for the list of view objects that have integration enabled.</td>
</tr>
<tr>
<td>End User</td>
<td>The user who generates transactions using Oracle Fusion applications with Oracle Social Network and who is saving the view object row in the product UI.</td>
<td>• Triggering the send of the view object row as a Social Object to Oracle Social Network.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Launching and using Oracle Social Network to view the Social Object.</td>
</tr>
</tbody>
</table>

At different times during the development process, a person must be able to perform several of these roles. For example, an Oracle Fusion Applications developer who is performing development through to testing will need to wear the hats for most or all of these roles to ensure that their integration pieces work correctly to send the data to Oracle Social Network.
67.2.1 How the Oracle Fusion Applications Developer Role Works

For developers, there are several phases of the development process in which they need to perform setup and configuration of the Oracle Social Network integration with their products.

- **Design Time** - Making a Business Object available for Oracle Social Network integration.
  
  The prerequisite is that the JDeveloper environment has at minimum the "Applications Core" extension loaded.

- **Configuring the Business Object for Oracle Social Network Integration**
  
  When implementing the Oracle Fusion Applications-Oracle Social Network integration provided by Oracle Fusion Middleware Extensions for Applications, the product team must already have a view object that meets the following criteria:
  
  - The view object is underpinned by one or more entity objects that are derived from the OAViewObjectImpl and OAEntityObjectImpl respectively.
  - Any rows from the view object must be derived from OAVewRowImpl.
  - The view object must not be "Expert" (SQL) based.
  
  Any view object that matches these criteria can be considered for Oracle Social Network integration availability. When a view object has been made available for Oracle Social Network integration, it can be enabled on the site using the "Manage Social Network Objects" page.

To set up a view object for Oracle Social Network integration, there is a common set of steps to provide basic integration. This basic integration can be customized based upon how the integration is to work for that view object. Customizations that can be applied are:

- Adding translated labels.
- Changing Oracle Social Network field data types, object title, object key, view details URL and registration name.

67.2.1.1 Configure the Business Object for Oracle Social Network Integration

The business object is underpinned by one or more entity objects that are derived from the OAVewObjectImpl and OAEntityObjectImpl respectively.

A wizard helps make view objects available for Oracle Social Network integration. The wizard will guide you through selecting the view objects and the set of attributes within the view object that are available for a site to send through to Oracle Social Network. All view objects that are to be available for Oracle Social Network integration must follow the steps listed here. Then they can have additional configuration made to tailor the integration further. The required steps to have a basic integration with Oracle Social Network are:

1. In JDeveloper’s Navigator pane, find the application module that contains the view object to be made available for Oracle Social Network integration.

2. Right-click the application module and select the Add OSN View Instances menu item. This will start the OSN Enabled View Instances wizard, shown in Figure 67-1.
At the top of the wizard is a dropdown list called Module. Choose the module that best represents the line of business for the current project.

Just beneath the Modules, there is a list of available view objects and a data model list. Find the view objects that are to be made available for Oracle Social Network integration from the list of available view objects and shuttle them into the data model. This signifies that the view object will be available for Oracle Social Network integration.

Click the view object in the data model pane to show the list of attributes for that view object in the Attributes tab.

Select the Enable check box next to any attribute that is to be available within that view object for Oracle Social Network integration. Only attributes that are enabled will be allowed to be sent to Oracle Social Network.

Click OK to finish the wizard.

If the deployment profile for the project is an ADF Library JAR file, then the META-INF folder and the fnd-osn-integration.xml file automatically will be bundled into the JAR file at the correct location. However, if the deployment profile is a regular JAR file, then the META-INF folder will need to be added as Project Output Contributor and ensure that the fnd-osn-integration.xml file is selected for inclusion in the JAR file. There should be no additional steps required for product integrators or special build steps so long as the META-INF/fnd-osn-integration.xml file is included in the root of the JAR file.

When the Model project is deployed and loaded at the client site, the view objects that were moved to the Data Model pane will be available in "Manage Social Network Objects," allowing them to be Oracle Social Network-enabled.
What happens when using the "OSN Enabled View Instances" wizard

After clicking OK to close the wizard, several tasks are performed with new artifacts generated. The first thing that happens is that a new file is created (if it did not already exist) called fnd-osn-integration.xml under the META-INF folder (that also will be created if it did not exist) of the JDeveloper project’s Application Source. There will only be one of these files per JDeveloper project and it will be re-loaded if the developer re-opens the wizard. This XML file contains the module information and the names of the view objects that have been made available for Oracle Social Network integration. A sample fnd-osn-integration.xml file looks like:

```xml
<?xml version="1.0" encoding="utf-8"?>
<fnd-osn-metadata xmlns="http://xmlns.oracle.com/apps/fnd/applcore/osn"
moduleId="84944D4BE32F441AE0401490D3C52A40">
  < AppModule Name="ProjectTaskAM"
    ApplicationModuleName="oracle.apps.fnd.applcore.demo.applicationModule.ProjectTaskAM" >
    <ViewUsage Name="Projects"
      ViewObjectName="oracle.apps.fnd.applcore.crmdemo.model.ProjectsVO"/>
    <ViewUsage Name="Tasks"
      ViewObjectName="oracle.apps.fnd.applcore.crmdemo.model.TasksVO">
      <ViewLinkUsage Name="ProjectsToTasksVL"
        ViewLinkObjectName="oracle.apps.fnd.applcore.crmdemo.model.ProjectsToTasksVL"
        SrcViewUsageName="oracle.apps.fnd.applcore.crmdemo.model.applicationModule.ProjectsTasksAM.Projects"
        DstViewUsageName="oracle.apps.fnd.applcore.crmdemo.model.applicationModule.ProjectsTasksAM.Tasks"
        Reversed="false"/>
      </ViewUsage>
    </ AppModule>
  </fnd-osn-metadata>
```

When deployed in an ADF Library JAR file, this file will be located at META-INF/fnd-osn-integration.xml allowing it be looked up directly on the class path from the "Manage Social Network Objects" page.

The other change that the wizard makes is to modify the view object definition XML file for the attributes that are available for Oracle Social Network integration. It adds or updates a new custom property called fnd:OSN_ENABLED_ATTR within the attribute XML node with a value of true to make that attribute available.

```xml
<ViewAttribute Name="DeptName">
  <Properties>
    <SchemaBasedProperties>
      <fnd:OSN_ENABLED_ATTR Value="true"/>
    </SchemaBasedProperties>
  </Properties>
</ViewAttribute>
```

67.2.1.2 Translating Oracle Social Network Labels

The view object and its attributes are displayed to the functional administrator and end users in the "Manage Social Network Objects" page and also within Oracle Social Network, so these strings must be translated. A functional administrator can use the "Manage Social Network Objects" taskflow to enable Oracle Social Network integration by selecting view objects and attributes. When Oracle Social Network displays a view object Social Object, the data is displayed as name-value pairs. The
Oracle Social Network integration re-uses the standard view object process for defining prompts for the translations that are sent to Oracle Social Network. This is done using the UI Hints properties that are already defined upon the business objects.

These UI Hint properties are used from the view object for translations:

- **Display Name** - used to show when a label is required for the view object.
- **Display Name (Plural)** - used to show when a plural label is required for the view object.
- **Description** - used for showing the description in the "Manage Social Network Objects" page.

These UI Hint properties are used from the view object’s attributes as Oracle Social Network field label translations:

- **Label** - used to show when a label is required for the attribute.
- **Display Name (Plural)** - used to show when a plural label is required for the attribute.

What happens when translated Oracle Social Network labels are provided

The view object definition XML file will be modified with these custom properties set:

- **View Object Property node names:**
  - LABEL - Display Name
  - LABEL_PLURAL - Display Name (Plural)
  - TOOLTIP - Description

- **Attribute Property node names:**
  - LABEL - Label
  - LABEL_PLURAL - Display Name (Plural)

The values provided in the Edit property window will be seeded into the translation bundle and the key will be stored as the ResId attribute of the relevant property node in the view object definition file. These are sample changes to the view object definition file:

```xml
<ViewObject Name="SndDemoDeptVO">
  <Properties>
    <SchemaBasedProperties>
      <LABEL ResId="DEMO_DEPARTMENT"/>
      <LABEL_PLURAL ResId="DEMO_DEPARTMENTS"/>
      <TOOLTIP ResId="DEMONSTRATION_DEPARTMENT_FOR_S"/>
    </SchemaBasedProperties>
  </Properties>
  <ViewAttribute Name="DeptName">
    <Properties>
      <SchemaBasedProperties>
        <LABEL ResId="DEPARTMENT_NAME"/>
        <LABEL_PLURAL ResId="DEPARTMENT_NAMES"/>
      </SchemaBasedProperties>
    </Properties>
  </ViewAttribute>
</ViewObject>
```
67.2.1.3 Configuring Oracle Social Network Field Data Types

Oracle Social Network integration is restricted to sending data using a limited number of data types. The view object attributes can be configured to provide some contextual information about how the attribute value will be shown as a field value in Oracle Social Network. For example, a string representing a URL will be kept in the attribute as a String but can be configured to show in Oracle Social Network to display as a clickable link rather than as a static String.

The supported data types, explanations, and their usage are listed in Table 67-2.

Note: A Gadget Type is a type definition for data sent to Oracle Social Network that allows the Oracle Social Network server to provide additional functionality when rendering the value in the UI. For example, a Percent gadget type can be displayed with a % symbol after it and must be a value between 0.0 and 100.0.

Table 67-2    Supported Data Types

<table>
<thead>
<tr>
<th>Gadget Type</th>
<th>Description</th>
<th>When to Use</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Allows the value sent to Oracle Social Network to be treated like a boolean. If the gadget that the conversation/wall is using has been modified in the gadget editor to show the editable representation of the field, then this will be displayed as a check box.</td>
<td>If the attribute being sent to Oracle Social Network is to be viewed as a boolean value.</td>
<td>If the Java Type of the attribute is anything other than java.lang.Boolean, then refer to the Boolean entry in Special Cases - Boolean.</td>
</tr>
</tbody>
</table>
### Table 67-2 (Cont.) Supported Data Types

<table>
<thead>
<tr>
<th>Gadget Type</th>
<th>Description</th>
<th>When to Use</th>
<th>Notes</th>
</tr>
</thead>
</table>
| Currency    | When sending a numeric attribute value representing a value in a currency to Oracle Social Network, this gadget type can send the additional currency code information relating to that attribute. To send the currency code with the numeric value perform the following steps:  
  - Set the Gadget Type on the numeric value attribute to "currency".  
  - Set the Currency Code Attribute property to be the name of another attribute in the view object whose value at runtime will be the Currency Code for the numeric value attribute. For example if the following attributes exist:  
    Attribute #1: "Credit"  
    Gadget Type="CURRENCY"  
    OSN_ENABLED="true"  
    OSN_CURRENCY_CODE_ATTR="CreditCurrencyCode"  
    Value="23"  
    Attribute #2:  
    "CreditCurrencyCode"  
    Gadget Type="STRING"  
    OSN_ENABLED="false"  
    Value="USD"  
    When sent to Oracle Social Network, the credit field will show as Credit: 23 USD | If the numeric value being sent to Oracle Social Network specifies a currency value and the currency code is also required to be sent to Oracle Social Network. | - The attribute for currency code should be a String value.  
- The attribute for currency code does not need to be flagged as OSN_ENABLED="true" as the currency code value will be sent as part of the numeric value payload being sent to Oracle Social Network.  
- If the attribute for currency code is marked as OSN_ENABLED="true" then it will be shown in a separate Oracle Social Network field from the currency code and if it also is flagged as the "Currency Code Attribute" property value on the numeric value attribute then it will be shown twice: in its own field and also as the numeric attribute’s currency code. |
| Date        | Display the attribute value as a formatted date. | If the attribute value is defined as a Date and can be shown as such in Oracle Social Network. | |
| Double      | Display a number as a floating point with a decimal fraction representation of the number. | When the attribute value will have a decimal fraction to be displayed. | |
### Table 67-2  (Cont.) Supported Data Types

<table>
<thead>
<tr>
<th>Gadget Type</th>
<th>Description</th>
<th>When to Use</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>Display a number as a whole number.</td>
<td>When the attribute value will not have a decimal fraction to be displayed.</td>
<td></td>
</tr>
<tr>
<td>Percent</td>
<td>Display a number as a percentage.</td>
<td>When the attribute value is to be displayed as a percentage.</td>
<td>The values can only be numbers between 0.0 and 100.0.</td>
</tr>
<tr>
<td>Person</td>
<td>Display the attribute as a clickable user who exists in Oracle Social Network. By sending the username or identifiable value for a user (most likely an email address in an LDAP-based Oracle Social Network) as a Person Gadget Type, Oracle Social Network will look up the display name for the user in its user security store (local/LDAP) and display a link using the user's display name. When the link is clicked, it will redirect to the Wall for the user represented by the person attribute.</td>
<td>When the attribute value represents the username/identifiable value for a user in Oracle Social Network. Only email addresses are supported.</td>
<td>• Attribute value must be a String, • Attribute value must be the identifiable value for a user in the security store being used by Oracle Social Network (local/LDAP).</td>
</tr>
<tr>
<td>String</td>
<td>Display a value.</td>
<td>If the Java Type of the object being retrieved from the view object does not conform to any of the other Gadget Types.</td>
<td></td>
</tr>
<tr>
<td>URL</td>
<td>Display a string value as a clickable URL link in Oracle Social Network.</td>
<td>If the attribute value is a fully qualified URL that an Oracle Social Network user will be allowed to click.</td>
<td>• The URL rendered in Oracle Social Network will not have a label appear next to it. • The URL value should be a fully qualified URL to the destination.</td>
</tr>
</tbody>
</table>

If the `OSN_GADGET_TYPE` is not provided (default state for the property), then the Oracle Social Network integration code will use the Java Type to Gadget Type mapping shown in Table 67-3 for sending the data to Oracle Social Network.
### Table 67-3  Java Type to Gadget Type Mapping

<table>
<thead>
<tr>
<th>ADFBC /Java Type</th>
<th>Gadget Type</th>
<th>Gadget Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.sql.Date</td>
<td>Date</td>
<td>Display the attribute value as a formatted date in Oracle Social Network.</td>
</tr>
<tr>
<td>java.sql.Timestamp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.util.Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oracle.jbo.domain.Date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oracle.jbo.domain.Timestamp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.lang.Integer</td>
<td>Integer</td>
<td>Display a number in Oracle Social Network as a whole number.</td>
</tr>
<tr>
<td>java.lang.Long</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.lang.Short</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.math.BigInteger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.util.concurrent.atomic.AtomicInteger</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.util.concurrent.atomic.AtomicLong</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.lang.Double</td>
<td>Double</td>
<td>Display a number in Oracle Social Network as a floating point with a decimal fraction representation of the number.</td>
</tr>
<tr>
<td>java.lang.Float</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.lang.Number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.math.BigDecimal</td>
<td></td>
<td></td>
</tr>
<tr>
<td>oracle.jbo.domain.Number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>java.lang.Boolean</td>
<td>Boolean</td>
<td>Allows the value sent to Oracle Social Network to be treated like a boolean. If the gadget that the conversation/wall is using has been modified (in the Oracle Social Network gadget editor) to show the editable representation of the field then this will be displayed as a check box.</td>
</tr>
<tr>
<td>any object using toString</td>
<td>String</td>
<td>Display a value.</td>
</tr>
<tr>
<td>any number supported by the Integer or Double Gadget Types</td>
<td>Currency</td>
<td>When sending an numeric attribute value representing a value in a currency to Oracle Social Network this gadget type allows for the sending of the additional currency code information relating to that attribute.</td>
</tr>
<tr>
<td>any number supported by the Integer or Double Gadget Types</td>
<td>Percent</td>
<td>Display a number as a percentage.</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>Person</td>
<td>Display the attribute in Oracle Social Network as a clickable user who exists in Oracle Social Network. By sending the username or identifiable value for a user, most likely email address in an LDAP based Oracle Social Network, as a Person Gadget Type Oracle Social Network will look up the display name for the user in its user security store (local/LDAP) and display a link with using the users display name. When the link is clicked it will redirect to the Wall for the user represented by the person attribute.</td>
</tr>
</tbody>
</table>
Table 67-3  (Cont.) Java Type to Gadget Type Mapping

<table>
<thead>
<tr>
<th>ADFBC /Java Type</th>
<th>Gadget Type</th>
<th>Gadget Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang.String</td>
<td>URL</td>
<td>Display a string value as a clickable URL link in Oracle Social Network.</td>
</tr>
</tbody>
</table>

To configure the attributes for the Oracle Social Network field gadget types (non-Currency):

- Open the view object.
- Open the Attributes tab and click an Attribute.
- In the Property Inspector, open the Applications - Oracle Social Network section.
- Change the value in the Oracle Social Network Gadget Type to the required type.
- Click Save to keep the changes.

For setting an attribute to the Currency gadget type, an additional step must be followed to add the currency code for the currency value. By setting the Currency Code attribute property for the attribute, when the Social Object is sent to Oracle Social Network, the currency code value also will be sent with the currency value so that Oracle Social Network can display both values when it displays the currency value. Note that the currency code attribute does not need to be available to Oracle Social Network integration for its value to be used when sending the currency so long as the attribute for the Currency is available to Oracle Social Network integration. These are the steps to set up an attribute for currency:

- Open the view object.
- Open the Attributes tab and click an attribute that is to have the gadget type of Currency.
- In the Property Inspector, open the Applications - Oracle Social Network section.
- Change the Oracle Social Network Gadget Type to Currency.
- In the Currency Code attribute property, enter the name of the attribute whose value will contain the currency code value.
- Click Save to keep the changes.

What happens when setting the Gadget Type

The View Attribute node in the view object definition XML file will be modified with these modified custom properties:

- fnl:OSN_GADGET_TYPE - the Oracle Social Network Gadget Type.
- fnl:OSN_CURRENCY_CODE_ATTR - the name of the attribute on the view object whose value will contain the currency code when fnl:OSN_GADGET_TYPE=currency.

For non-currency attributes:

```
<ViewAttribute Name="ColBoolean">
<Properties>
```
<SchemaBasedProperties>
  <fnd:OSN_ENABLED_ATTR Value="true"/>
  <fnd:OSN_GADGET_TYPE Value="boolean"/>
</SchemaBasedProperties>
</Properties>
</ViewAttribute>

For currency attributes:

<ViewAttribute Name="ColCurrency">
  <Properties>
    <SchemaBasedProperties>
      <fnd:OSN_ENABLED_ATTR Value="true"/>
      <fnd:OSN_GADGET_TYPE Value="currency"/>
      <fnd:OSN_CURRENCY_CODE_ATTR Value="ColCurrencyCode"/>
    </SchemaBasedProperties>
  </Properties>
</ViewAttribute>
<ViewAttribute Name="ColCurrencyCode"/>

### Special Cases - Boolean

If an attribute represents a boolean value, a conversion will be required to send the value to Oracle Social Network. The recommended way to perform a Boolean conversion is to use the Attribute Conversion Framework, described in [Implementing the Integration Attribute Conversion Framework](#). If the conversion framework cannot be used, then below are the alternative instructions on how to use a transient to achieve the conversion.

The Oracle Social Network integration requires that any attribute marked as Gadget Type Boolean must provide the string value "T" to represent true or "F" to represent false. As part of the Oracle Social Network integration, if a Java Type of `java.lang.Boolean` is provided, it implicitly is sent to Oracle Social Network as "T" or "F."

An additional exception was made that if an attribute was of type String with Gadget Type of Boolean and the value provided being "T" or "F," then that value will be sent to Oracle Social Network. Any other values or types will not be sent to Oracle Social Network with a message being written to the Diagnostic log.

In Oracle Fusion Applications, it is common practice to define a Boolean value as a String and use values such as "Y" and "N" or "1" and "0." However, it is not just restricted to those values. In this case, just setting the Gadget Type to Boolean will not send any value to Oracle Social Network.

Follow these alternative instructions on how to use a transient to achieve the conversion if the conversion framework cannot be used:

- Add a new transient attribute to the view object.
- Set the Type of the attribute to be Boolean.
- Implement the conversion between the string boolean attribute and the Java Boolean attribute as a Groovy expression or as a Java override of the getter attribute of the transient attribute in the RowImpl class for the view object. For example, this Groovy expression can be used in the "Y"/"N" case:

  ```groovy
  "Y".equalsIgnoreCase(<your attribute name>)
  ```

- In the **Dependencies** page of the attribute wizard, make the Boolean transient attribute dependent upon the String boolean attribute.
• Set the Gadget Type for the Boolean transient attribute as boolean.

• Either re-open the OSN Enabled View Instances wizard, find the view object and click the Enabled check box for the transient attribute and click OK.

  or

  Open the Attributes tab in the view object, select the transient attribute, go to the Property Inspector and change the OSN Enabled property to be "true."

• Click Save to keep the changes.

Special Cases - Lookups and Conversion Values

Attributes that are a lookup code must provide a meaning or translated value to Oracle Social Network. The preferred method of conversion for lookups is to use the lookup converters in the conversion framework, as described in Implementing the Integration Attribute Conversion Framework.

The primary use case is lists of values where the lookup code is stored in the view object attribute and a lookup list of values provides a mapping between the lookup code and the descriptive meaning. Supporting the lookups in Oracle Social Network integration requires creating a new attribute that contains the lookup display value. There are two ways the attribute can be defined to provide the meaning value.

• Foreign key lookup

  Most view objects with a lookup code will already have a join to get the lookup display value. Essentially, a foreign key is defined through the lookup entity object in the view object definition. In this case, the meaning attribute can be Oracle Social Network enabled by using one of these methods:

  – Open the Add OSN View Instances wizard, find the view object, select the Enabled check box for the meaning attribute, and click OK.

  – Open the Attributes tab in the view object, select the meaning attribute, go the Property Inspector and change the OSN Enabled property to be "true".

• Transient attribute lookup

  A string transient attribute will be defined and the view object Row Impl’s getter for the transient will perform the lookup based upon the lookup code and return the meaning value. This is similar to how boolean values are handled where the transient attribute will be Oracle Social Network enabled.

Special Cases - Timezone Affected Values

The function of Oracle Social Network requires that a date value be passed to Oracle Social Network as the number of milliseconds since Epoch. The Oracle Social Network integration takes the date value provided from the attribute count in milliseconds and sends it to Oracle Social Network. Oracle Social Network then will display that value with the settings configured with Oracle Social Network. If there must be a deviation to how this works to account for timezoned dates, then a transient attribute approach where the time is modified and passed to Oracle Social Network integration as a new date will need to be performed.

• Add a new transient attribute to the view object.

• Set the Type of the attribute to be Date/TimeStamp.
• Implement the timezone conversion between the original Date/Time Stamp attribute and the Date/Time Stamp transient attribute so that the value will be correct when being retrieved as milliseconds since Epoch as a Groovy expression or as a Java override of the getter attribute of the transient attribute in the RowImpl class for the view object.

• In the Dependencies page of the attribute wizard, make the Date/Time Stamp transient attribute dependent upon the Original Date/Time Stamp attribute.

• Set the Gadget Type for the transient attribute as Date.

• Either re-open the Add OSN View Instances wizard, find the view object and select the Enabled check box for the transient attribute and click OK.

  or

  Open the Attributes tab in the view object, select the transient attribute, go to the Property Inspector and change the OSN Enabled property to be "true".

• Click Save to keep the changes.

Special Cases - Other types of attributes that do not fit the defined list of Oracle Social Network Gadget Types

If there are other value types in a view object attribute that need to be sent to Oracle Social Network or provided in a different format, the preferred method to convert other types is a custom conversion as described in How Custom Attribute Conversions Are Implemented. If the conversion framework cannot be used, then follow these alternative instructions on how to use a transient to achieve the conversion.

• Add a new transient attribute to the view object.

• Set the Type of the attribute to be String.

• Implement the conversion between the custom value type and the String transient attribute as a Groovy expression or as a Java override of the getter attribute of the transient attribute in the RowImpl class for the view object.

• In the Dependencies page of the attribute wizard, make the String transient attribute dependent upon the custom value type attribute.

• Set the Gadget Type for the transient attribute as String.

• Either re-open the Add OSN View Instances wizard, find the view object, select the Enabled check box for the transient attribute, and click OK.

  or

  Open the Attributes tab in the view object, select the transient attribute, go to the Property Inspector and change the OSN Enabled property to be "true".

• Click Save to keep the changes.

67.2.1.4 Implement the Integration Attribute Conversion Framework

An Oracle Fusion Applications developer can choose attributes from a view object that can be sent to Oracle Social Network. However, in some circumstances, the value that is retrieved may not be the value that is required to be sent. In these cases, a converter is required to take the attribute value from the view object row and change it into the value required. The converter is defined against the view object's attribute definition.
and executed automatically by the Oracle Social Network integration code. The converters:

- Allow being able to send to Oracle Social Network a modified version of the value stored in Oracle Fusion Applications.
- Provide the ability for lookup values (such as LOV values) to be looked up without having to add additional transient attributes to the original view object.
- Ensure the modification tracking of updates will work correctly as the modification is made to the "code" value in a lookup, not the "display" value. Previous to this enhancement, the "display" value would be required to be Oracle Social Network-enabled rather than the "code".

For example, Oracle Fusion Applications developers configure the view objects and attributes for the Oracle Social Network integration. However, restrictions on the Oracle Social Network server, and product team requirements mean that the value stored in the database table for a record cannot be sent to Oracle Social Network as-is. In these cases, a converter is required to be assigned to the attribute definition to take the stored value and convert it into the required format or value required by Oracle Social Network without any additional effect on the send to Oracle Social Network process.

There are four out-of-the-box converters ready to be used. A single converter can be added to the Attribute by using the values in the OSN Converter property in the property inspector. The OSN Converter property has a dropdown list containing these out-of-the-box converters. If the converter requires additional values to be supplied before the conversion can occur, then these can be set in the OSN Converter Parameters property that resides immediately below the OSN Converter property. The list of additional converter parameters to fill in can be seen by choosing the Edit... option from the property menu. The four out-of-the-box converters are:

- **Boolean Converter**
  
  Provides the ability to convert the object passed in into a boolean type based upon a customizable list of accepted (whitelist/true) or rejected (blacklist/false) values. If the object passed into `convertBoolean()` is of type `java.lang.Boolean`, then this will be returned immediately. Otherwise, `toString()` will be called on the object and the resultant String will be compared using a case-insensitive comparison to the `trueValues` or `falseValues` using the following algorithm:

  - `trueValues` and `falseValues` provided, then use `trueValues` and ignore `falseValues`.
  
  - `trueValues=null` and `falseValues=null`, default `trueValues` to be "y,t,1,yes,true".
  
  - `trueValues` supplied, then split the `trueValues` CSV and see if the passed-in value matches one of the values. If so, return true, otherwise false.
  
  - `falseValues` supplied and `trueValues=null`, then split the `falseValues` CSV and see if the passed-in value matches one of those values. If so, return false, otherwise true.

  The CSV parameters will conform to a standard CSV encoding to ensure the parameters do not become corrupt during persistence. Double quotes will be escaped by a second double quote (for example, " becomes """) and each entry in the CSV can be quoted.
Note: This converter is strictly meant for converting "code values" into boolean values. Looked-up translated values are not supported with the OsnBooleanConverter. If a translated value is required to be sent to Oracle Social Network, either the translation can be sent as a string (using the LovConverter or LookupConverter) or a custom converter will need to be written. When mentioning the default trueValues whitelist, the values of yes/true are constant code values, not translatable versions of yes/true.

OSN Converter: BooleanConv

Parameters:
- trueValues - (optional) - the CSV string list of values that when matched will return true from the converter.
- falseValues - (optional) - the CSV string list of values that when matched will return false from the converter.

Validates For: If attribute value to convert is null.

Upon Validation Failure Return: null

Can Convert Null: No

Example:

ColBoolean field in Oracle Social Network will be true if the attribute value is either "y" or "yes" (case-insensitive). All other values will return false.

```
<ViewAttribute
  Name="ColBoolean"
  PrecisionRule="true"
  EntityAttrName="ColBoolean"
  EntityUsage="FndDemoOsnFieldsEO"
  AliasName="COL_BOOLEAN">
  <Properties>
    <SchemaBasedProperties>
      <fnd:OSN_ENABLED_ATTR
        Value="true"/>
      <fnd:OSN_GADGET_TYPE
        Value="boolean"/>
      <fnd:OSN_CONVERTER
        Value="BooleanConv"/>
      <fnd:OSN_CONVERTER_PARAM
        Value="trueValues=y,yes"/>
    </SchemaBasedProperties>
  </Properties>
</ViewAttribute>
```

- Percentage Converter - 0.0-1.0 range converted to 0.0-100.0 range

Converts the value between 0.0 and 1.0 that is provided and returns the matching percentage value between 0.0 and 100.0 required by Oracle Social Network for the percent Gadget Type. The convertPercentage method takes the value supplied, ensures that it meets the exception statement, converts it to a double value, and multiplies it by 100.0.

OSN Converter: ZeroToOnePercentConv

Parameters: None

Validates For: A null, a non-numeric value, or a value not between 0.0 and 1.0 is passed in.
Upon Validation Failure Return: null
Can Convert Null: No

Example:

```xml
<ViewAttribute
    Name="ColPercent"
    PrecisionRule="true"
    EntityAttrName="ColPercent"
    EntityUsage="FndDemoOsnFieldsEO"
    AliasName="COL_PERCENT">
    <Properties>
      <SchemaBasedProperties>
        <fnd:OSN_ENABLED_ATTR
          Value="true"/>
        <fnd:OSN_GADGET_TYPE
          Value="percent"/>
        <fnd:OSN_CONVERTER
          Value="ZeroToOnePercentConv"/>
      </SchemaBasedProperties>
    </Properties>
</ViewAttribute>
```

• **Attribute Lookup Converter**

  The converter will look up another attribute value from the same view object row and substitute it for the actual attribute value.

  **OSN Converter: OsnFieldValueConverter**

  **Parameters:**
  
  displayAttributeName - mandatory - the name of another attribute in the view object to return the value of.

  **Validates For:** A null or a non-existent display attribute name supplied.

  Upon Validation Failure Return: null
Can Convert Null: No

**Example:**

The value assigned to the **ColString** attribute will be sent to Oracle Social Network as the value for the **ColLookupString** Oracle Social Network field.

```xml
<ViewAttribute
    Name="ColLookupString"
    PrecisionRule="true"
    EntityAttrName="ColLookupString"
    EntityUsage="FndDemoOsnFieldsEO"
    AliasName="COL_LOOKUP_STRING">
    <Properties>
      <SchemaBasedProperties>
        <fnd:OSN_ENABLED_ATTR
          Value="true"/>
        <fnd:OSN_CONVERTER
          Value="LookupConv"/>
        <fnd:OSN_CONVERTER_PARAM
          Value="displayAttributeName=ColString"/>
        <LABEL
          ResId="LOOKUP_CONVERTED"/>
      </SchemaBasedProperties>
    </Properties>
</ViewAttribute>
```
• **LOV Converter**

If an attribute has an LOV associated with it, then the LOV converter will use the definition of the attribute and look up the LOV associated with the attribute. It can then look up the display value from the LOV that is associated with the attribute value and send the LOV display value to Oracle Social Network.

**OSN Converter: OsnAttributeLookupLovConverter**

**Parameters:** None

**Validates For:** An LOV List Binding must be defined for the attribute.

**Upon Validation Failure Return:** null

**Can Convert Null:** No

**Example:**

In this example the display value from the LOV_ColStatus LOV will be sent that matches the value of this attribute.

```xml
<ViewAttribute
Name="ColStatus"
PrecisionRule="true"
EntityAttrName="ColStatus"
EntityUsage="FndDemoOsnFieldsEO"
AliasName="COL_STATUS">
<Properties>
<SchemaBasedProperties>
<fnd:OSN_CONVERTER Value="LovConv"/>
</Properties>
</ViewAttribute>
```
67.2.1.4.1 How Custom Attribute Conversions Are Implemented

If the conversions supplied do not meet the requirement for a conversion of an attribute value being sent to Oracle Social Network, then a custom converter can be written. The conversion framework converters are based around a central Java class called OsnFieldValueConverter (oracle.apps.fnd.appcore.oaext.osn.publicutil.converter). It is the abstract base class that all converters will inherit. There must be a zero parameter default constructor that will be used to create the instance of the converter. It contains a solitary public method, run, that will be passed the value to convert and a context object comprised of the attribute name, the intended Oracle Social Network gadget type, and the ViewRowImpl containing the value to convert.

The run method is responsible for executing the conversion by calling the abstract convert method and not allowing any exceptions that happen during conversion to be bubbled outside of the converter. If an exception occurs during the conversion, the implementing converter can choose to return the passed-in value, or null (the default behavior) back from the conversion.

It also has logging around the convert call to describe the performance and the converted result of the conversion attempt. A short circuit is built in for null handling using the canConvertNull override method. If a converter returns false for this method (which is the default implementation), then the convert method will not be called and null immediately will be returned from the run method. If a converter can convert null into another value, then the converter can override this method and return true so that the convert method will always be called.

Any converter that is built as a sub-class of OsnFieldValueConverter must implement the convert method that has the same signature as the run method, except that it can throw a checked OsnConversionException. This exception can provide details from the converter implementation about why a conversion attempt failed.

To provide more type checking abilities during conversion, three abstract "base" converters, which implement the OsnFieldValueConverter, are available that implement the convert method but call another abstract method that any sub-classes must implement that will ensure that the return value from a conversion is of the type required:

- **OsnBaseBooleanValueConverter** - Has an abstract convertBoolean method that takes the same parameters as convert but requires that a Java boolean native type be returned. Since the Oracle Social Network server requires that a boolean be provided as either "T" or "F", the boolean result from convertBoolean will be returned from the convert method as "T"/"F".

- **OsnBasePercentageValueConverter** - Has an abstract convertPercentage method that takes the same parameters as convert but requires that a Java double native type be returned. It also will do post processing to ensure that the value conforms with the expected range of 0.0 to 100.0 from Oracle Social Network and throw an OsnConversionException if the converted value from convertPercent is out of range.

- **OsnBaseDateValueConverter** - Has an abstract convertDate method that takes the same parameters as convert but requires that a Java long native type be returned. The long value signifies the Oracle Social Network requirement that all Date gadget type values be represented as the number of milliseconds since Epoch.
For an Oracle Fusion Applications product team to consume a converter for its attributes, the fully-qualified class name for the converter can be manually typed into the OSN Converter property (fnd:OSN_CONVERTER). If there are custom parameter values required for the converter, they will be read from the view object Attributes Converter Parameters parameter. The fnd:OSN_CONVERTER_PARAM is in the format "parameter1=value1;parameter2=value2". Each parameter will need to have its own getter and setter on the converter class. In this example, the setParameter1 method would be called with value1 and the setParameter2 method would be called with value2 before the run method was called. The getter for these parameters will need to be annotated with @OsnConverterParam to identify it as a converter parameter and show in the OSN Converter Parameters property menu UI. Any parameter values that are supplied in fnd:OSN_CONVERTER_PARAM that do not have a matching getter and setter will be ignored in the property menu. Expression Language expressions are not supported as the fnd:OSN_CONVERTER_PARAM, or as the values within the parameter list.

What happens at runtime when sending to OSN:

When the view object row attribute value is loaded and is about to be put into the payload to send to Oracle Social Network, the converter will be enacted using the following process:

- Read the Converter parameter (fnd:OSN_CONVERTER).
- Create the instance of the converter class using the fully-qualified class name.
- Read and set any parameters from fnd:OSN_CONVERTER_PARAM on the converter instance.
- Determine the run method called to execute the conversion.
- Put the value returned from the converter into the Oracle Social Network payload for that attribute.

### 67.2.1.5 Configuring Oracle Social Network Object Title

When a view object row is sent to Oracle Social Network as a Social Object, it will be displayed in the Oracle Social Network UI as a record with a title and a collection of name-value pairs. The value that is shown as the title of the Social Object can be configured on the view object. The following steps describe how to set the title for the Social Object:

- Open the view object in JDeveloper and click the **Attributes** tab.
- Determine that the attribute is to be used as the Social Object title value. If this is a pre-existing attribute value, skip the next three optional steps.
- [Optional-generated title] Define a new transient attribute that has a type of String.
- [Optional-generated title] Implement the title generation in the transient attribute as a Groovy expression or as a Java override of the getter attribute of the transient attribute in the RowImpl class for the view object.
- [Optional-generated title] In the **Dependencies** page of the attribute wizard, make the String transient attribute dependent upon the custom value type attribute.
- Select the attribute (transient or actual).
• Access the **Property Inspector** in the **Applications - Oracle Social Network** section and set the **Oracle Social Network User Key** to be "true".

• Click **Save** to keep the changes.

**What happens when you define the Oracle Social Network Social Object Title**

The Attribute definition within the view object definition XML file will be modified with the following custom property:

```xml
fnd:OSN_USER_KEY - The Oracle Social Network User Key custom attribute.
```

The following is an example of the modification in the view object attribute definition XML file:

```xml
<ViewAttribute Name="DeptName">
  <Properties>
    <SchemaBasedProperties>
      <fnd:OSN_ENABLED_ATTR Value="true"/>
      <fnd:OSN_USER_KEY Value="true"/>
    </SchemaBasedProperties>
  </Properties>
</ViewAttribute>
```

**Note:**

• There should be **only one** attribute with the Oracle Social Network User Key custom property value set to "true". If there are multiple attributes with the custom property set, then the first one that is found during processing will be used and all others will be **ignored**.

• If no attributes are provided with this custom property set, then the concatenated values of the attributes that comprise the view object primary key will be used.

**67.2.1.6 Configuring Oracle Social Network Object Key**

When a view object row is sent to Oracle Social Network, the Social Object will be assigned an internal unique identifier that will represent the Social Object during any interactions with Oracle Social Network. To perform subsequent updates to the Social Object from the view object, the Social Object also retains an external key based upon a single attribute value that must be sent to Oracle Social Network with every call. Without any additional configuration, the Oracle Social Network integration will build a concatenated value based upon the values of the attributes marked as the primary key that should suffice in most scenarios. If this concatenated value is longer than 256 characters, or the developer has a better key to use to uniquely identify the row, the Oracle Social Network integration provides the ability to override the automatically-generated value with a value of a custom property called OSN Object Key. For example, if a child object in a parent-child relationship is being sent to Oracle Social Network, then its primary key may be comprised of the parent object's primary key plus a unique value for the child. If the child value is unique within the child table, then the parent primary key is not required for the Social Object to uniquely identify with the view object row. So by marking the child unique value as the Oracle Social Network Object Key, there will be a slight performance gain as the external key for the Social Object does not need to be calculated.

To set up the Oracle Social Network Object Key:
Open the view object in JDeveloper and click the Attributes tab.

Determine that attribute is to be used as the Social Object external key value. If this is a pre-existing attribute value then skip the next three optional steps.

[Optional-generated key] Define a new transient attribute that has a type of String.

[Optional-generated key] Implement the key generation in the transient attribute as a Groovy expression or as a Java override of the getter attribute of the transient attribute in the RowImpl class for the view object.

[Optional-generated key] In the Dependencies page of the attribute wizard, make the String transient attribute dependent upon the custom value type attribute.

Select the attribute (transient or actual).

Access the Property Inspector in the Applications - Oracle Social Network section and set the Object Key to be "true."

Click Save to keep the changes.

**What happens when you define the Oracle Social Network Social Object Key**

The Attribute definition within the view object definition XML file will be modified with the following custom property:

```
fnd:OSN_OBJECT_KEY - The Object Key attribute.
```

The following is an example of the modification in the view object Attribute definition XML.

```
<ViewAttribute Name="DeptNum">  
 <Properties>  
   <SchemaBasedProperties>  
     <fnd:OSN_ENABLED_ATTR Value="true"/>  
     <fnd:OSN_OBJECT_KEY Value="true"/>  
   </SchemaBasedProperties>  
 </Properties>  
</ViewAttribute>
```

**Note:**

- Single attribute primary key view objects do not need to set this value because no concatenations need to occur to get the primary key value to use.

- There should only one attribute with the Oracle Social Network Object Key custom property value set to "true". If there are multiple attributes with the custom property set, then the first one that is found during processing will be used and all others will be ignored.

- If no attributes are provided with this custom property set, then the concatenated values of the attributes that comprise the view object primary key will be used.
67.2.1.7 Configuring the View Details URL

Use the information in this section to navigate from the Oracle Social Network Social Object into the Oracle Fusion Applications user interface.

After a view object row is sent to Oracle Social Network as a Social Object there may be a need for the user to modify the original values for that row. The mechanism that allows navigating from the Social Object back into the record in Oracle Fusion Applications uses a "View Details" URL that is shown when viewing the Social Object details. To be able to generate a URL that can navigate into Oracle Fusion Applications, there must be a taskflow to be called that can accept parameters to load the row in the product application page as well as configuration on the view object that provides the information allowing the Oracle Social Network integration to build the URL. The configuration on the view object is performed by setting the following properties:

- **Nav Web Application** (fnd:OSN_FA_WEBAPP) - This is a *mandatory* field. It is the identifier for a registered web application that is to be called back into. The value provided here must be seeded into Topology Manager as a deployed application. It is used in the Oracle Social Network integration to determine the web applications that will be called back into when the View Details link is clicked. This web application will be used to look up the base URL for the callback URL. An example is shown here:

```xml
<Properties>
  <SchemaBasedProperties>
    <fnd:OSN_FA_WEBAPP>
      Value="ApplcoreCRMDemo"/>
  </SchemaBasedProperties>
</Properties>
```

- **Nav View Id** (fnd:OSN_FA_VIEW_ID) - This is a *mandatory* field. This is the identifier for the view that will be displayed upon callback. It is used in the Oracle Social Network integration to determine what view is to be loaded when the callback is made from the View Details link. An example is shown here:

```xml
<Properties>
  <SchemaBasedProperties>
    <fnd:OSN_FA_VIEW_ID>
      Value="MyPage"/>
  </SchemaBasedProperties>
</Properties>
```

- **Nav Page Params List** (fnd:OSN_FA_PAGE_PARAMS_LIST) - This is a semicolon separated list of name-value pairs to be passed to the Oracle Fusion application's page as page level parameters when the callback is made from the View Details link. An example is shown here:

```xml
<Properties>
  <SchemaBasedProperties>
    <fnd:OSN_FA_PAGE_PARAMS_LIST>
      Value="key1=value1;key2=value2"/>
  </SchemaBasedProperties>
</Properties>
```

- **Nav Taskflow Id** (fnd:OSN_FA_NAV_TASKFLOW_ID) - This is a *mandatory* field. This is an Id to a bound task flow available in the page. This task flow will be
called and passed parameters from `OSN_FA_NAV_TASK_PARAMS_LIST` when the callback is made from the View Details link. An example is shown here:

```xml
<Properties>
  <SchemaBasedProperties>
    <fnd:OSN_FA_NAV_TASKFLOW_ID Value="/WEB-INF/StartupDefaultMainTaskFlow.xml#StartupDefaultMainTaskFlow"/>
  </SchemaBasedProperties>
</Properties>
```

- **Nav Task Key List** (`fnd:OSN_FA_NAV_TASK_KEY_LIST`) - This is a semicolon separated list of name-value pairs to be passed to the page containing the task flow. These parameters are used to distinguish one instance of the task flow from any others that are also running. An example is shown here:

```xml
<Properties>
  <SchemaBasedProperties>
    <fnd:OSN_FA_NAV_TASK_KEY_LIST Value="key1=value1;key2=value2"/>
  </SchemaBasedProperties>
</Properties>
```

- **Nav Task Params List** (`fnd:OSN_FA_NAV_TASK_PARAMS_LIST`) - This is a semicolon separated list of name-value pairs to be passed to the called bound task flow in the Oracle Fusion application's page. These will be specific to the task flow and need to be defined as input parameters on the task flow. An example is shown here:

```xml
<Properties>
  <SchemaBasedProperties>
    <fnd:OSN_FA_NAV_TASK_PARAMS_LIST Value="key1=value1;key2=value2"/>
  </SchemaBasedProperties>
</Properties>
```

- **Task Label** (`fnd:OSN_FA_NAV_TASK_LABEL`) - This is the label to be shown on the tab displayed that contains the bound task flow. This label will be a key into the appropriate translated bundle. An example is shown here:

```xml
<SchemaBasedProperties>
  <fnd:OSN_FA_NAV_TASK_LABEL Value="SearchFlatTableVO1_OSN_FA_NAV_TASK_LABEL"/>
</SchemaBasedProperties>
```

```xml
<trans-unit id="SearchFlatTableVO1_OSN_FA_NAV_TASK_LABEL">
  <source>Flat Table</source>
  <target/>
</trans-unit>
```

- **Task Label Attribute** (`fnd:OSN_FA_NAV_TASK_LABEL_ATTR`) - This is the name of an attribute in the view object that contains the label to be shown on the tab displayed that contains the bound task flow. An example is shown here:

```xml
<Properties>
  <SchemaBasedProperties>
    <fnd:OSN_FA_NAV_TASK_LABEL_ATTR Value="TaskLabelAttribute"/>
  </SchemaBasedProperties>
</Properties>
```
Note:

- This should not be used if the Task Label property is set. If both are set, then this one will take precedence and the Task Label will be ignored.
- The attribute to which this property refers can be a transient or actual attribute.

The values supplied for Nav Page Params List, Nav Task Key List, Nav Task Params List and Task Label can contain attribute name tokens that will be resolved when the View Details URL is being generated before it is sent to Oracle Social Network. The token is made up of an attribute name contained within curly brackets {}.

For example, if the Nav Task Params List is defined as "deptNumParam={DeptNum}" then the value of the view object row Attribute called "DeptNum" will replace the {DeptNum} token.

To set up the view object for the View Details URL:

- Open the view object in JDeveloper and click the General tab.
- In the Property Inspector, go to the Applications - Oracle Social Network section.
- Fill in the following custom properties:
  - Nav Web Application
  - Nav View Id
  - Nav Page Params List
  - Nav Taskflow Id
  - Nav Task Key List
  - Nav Task Params List
  - Task Label
  - Task Label Attribute
- Click Save to keep the changes.

What happens when you configure the View Details URL Properties

The view object definition XML file will be modified with the following custom properties:

```xml
<ViewObject Name="SndDemoDeptVO">
  <Properties>
    <SchemaBasedProperties>
      <fnd:OSN_FA_WEBAPP Value="SocialNetworkDemo"/>
      <fnd:OSN_FA_VIEW_ID Value="SocialNetworkDemo"/>
      <fnd:OSN_FA_NAV_TASKFLOW_ID Value="/WEB-INF/DemoDepartmentTF.xml#DemoDepartmentTF"/>
      <fnd:OSN_FA_NAV_TASK_PARAMS_LIST Value="deptNumParam={DeptNum}"/>
      <fnd:OSN_FA_NAV_TASK_KEY_LIST Value="deptNumKeyParam={DeptNum}"/>
      <fnd:OSN_FA_PAGE_PARAMS_LIST Value="deptNumPageParam={DeptNum}"/>
      <fnd:OSN_FA_NAV_TASK_LABEL ResId="NAVIGATE_DEPT_DEPTNUM"/>
    </SchemaBasedProperties>
  </Properties>
</ViewObject>
```
When the end user submits the view object row this will trigger the Social Object to be generated for sending to Oracle Social Network. As part of this generation the View Details URL will be constructed and added to the Social Object payload. The URL is generated using an algorithm that will perform lookups, such as determining the base URL from the Web App Id, and other parameter manipulations that will convert these parameters into a string that UIShell will accept as secure and valid. The UIShell, when called with this URL, will be able to determine what the parameters were and how to load the page and taskflow. For example, the following parameters will generate into the URL.

- **View Object Custom Properties**
  
  &nbsp;&nbsp;&nbsp;&nbsp;fnd:OSN_FA_WEBAPP="SocialNetworkDemoApp"
  &nbsp;&nbsp;&nbsp;&nbsp;fnd:OSN_FA_VIEW_ID="SocialNetworkDemo"
  &nbsp;&nbsp;&nbsp;&nbsp;fnd:OSN_FA_NAV_TASKFLOW_ID="/WEB-INF/DemoDepartmentTF.xml#DemoDepartmentTF"
  &nbsp;&nbsp;&nbsp;&nbsp;fnd:OSN_FA_NAV_TASK_PARAMS_LIST="deptNumParam={DeptNum}"
  &nbsp;&nbsp;&nbsp;&nbsp;fnd:OSN_FA_NAV_TASK_LABEL="Navigated {DeptNum}Department"

- **Topology Manager Setup**
  

- **View Object Row**
  
  DeptNum attribute value=350

- **Generated View Details URL**
  
  http://localhost:7001/snd-context-root/faces/SocialNetworkDemo?fnd=%252FWEB-INF%252FDemoDepartmentTF.xml%2523DemoDepartmentTF%3BdeptNumParam%253D350%3B%3BNavigated%2B350%2BDpartment%3Bfalse%3B256

### 67.2.1.8 Changing the Oracle Social Network Registration Name

When a view object is registered with Oracle Social Network as a Social Object definition, the definition must have a unique name assigned to it that will be used to identify the definition. Without any additional configuration, the registration name will be the concatenation of the application module Id and the fully qualified view object Name. For example, if the application module Id was FND and the view object was oracle.apps.fnd.applcore.demo.FndDemoDeptVO, then the registration definition would be FND.oracle.apps.fnd.applcore.demo.FndDemoDeptVO. In addition to the registration name for the view object, the Social Object Field Names are based upon the names of the attributes in the view object.

The Oracle Social Network integration provides a way to override the view object Social Object registration name and the Social Object Field Names. This can be done for the continued support of the existing Oracle Social Network registrations (such as for CRM/HCM) or to ensure that the length of the names does not exceed the 256 character limit.

To set the overrides of the Social Object definition registration names:

- Open the view object in JDeveloper and click the General tab.
- In the Property Inspector go to the **Applications - Oracle Social Network** section.
- Set the following custom property to override the view object's Social Object registration definition name:
  
  fnd:OSN_DEFINITION_NAME - This must be in the format &lt;module id&gt;.&lt;name&gt;. For example: FND.FndDempDept.
• Click the Attributes tab and select the attribute that is to have its Social Object Field Name overridden.

• Set the following custom property:
  \[fnd:OSN_DEFINITION_NAME\] - The Social Object Field name the attribute will be referred to as.

• Click Save to keep the changes.

What happens when you change the Social Object Registration Name

The following custom properties are modified within the view object XML definition.

• \[fnd:OSN_DEFINITION_NAME\] - Definition Name
• \[fnd:OSN_ATTR_DEFINITION_NAME\] - Oracle Social Network Attribute Name Override

The view object definition XML file will be modified with these custom properties:

```xml
<ViewObject Name="FndDemoOsnFieldsVO">
  <Properties>
    <SchemaBasedProperties>
      <fnd:OSN_DEFINITION_NAME Value="fnd.FndDemoOsnFieldsVO"/>
    </SchemaBasedProperties>
  </Properties>
  <ViewAttribute Name="OsnTestRecordId">
    <Properties>
      <SchemaBasedProperties>
        <fnd:OSN_ENABLED_ATTR Value="true"/>
        <fnd:OSN_ATTR_DEFINITION_NAME Value="TestId"/>
      </SchemaBasedProperties>
    </Properties>
  </ViewAttribute>
</ViewObject>
```

67.2.1.9 Configuring View Objects that Have Multiple Updatable Entities

The Oracle Social Network integration can support view objects that are comprised of multiple updatable entity objects.

A multi-updatable entity object view object is:

• A view object that has multiple entity objects selected to be within it.

• The Join Type must be an inner join.

• The entity objects that are participating in the update must have an association defined between them that creates an accessor in the Primary entity object that returns an instance of an \[OAEntityImpl\] (or one of its sub-classes) objects that is the instance of the Secondary entity object.

There are several cases in Oracle Fusion Applications where an Oracle Social Network integration available view object is tied to multiple updatable entity objects. For example, several Oracle Fusion applications use TCA (HZ_PARTIES) to manage key attributes about a person or company. Many teams, such as Procurement and CRM, extend the TCA schema with their own tables and columns to meet the information gathering requirements for their product. Procurement owns an entity called Supplier that extends the TCA Party schema to maintain information about a company from which you would like to purchase goods and services. The Supplier view object would therefore have updateable attributes from both the Supplier entity object and the TCA Party entity object. ATG business event registration for Oracle Social Network...
integration typically only references the primary entity object. In the TCA case, a customer can go to the Maintain Supplier UI and update a single attribute, such as Supplier Name. The Supplier Name is maintained in the TCA Party entity object that, in this case, is not the primary entity object. We must publish these cases to Oracle Social Network as update events but we do not want to place the Oracle Social Network integration events on multiple entity objects to avoid mass duplication of events.

The solution for this use case is to register the Primary entity object (Supplier entity object) as a post transaction listener for the Secondary entity object (Party entity object). Whenever an update occurs in the Secondary entity object, the primary entity object will also be notified and its post transaction processing will send the view object data from both entity objects to Oracle Social Network.

These implementation steps are required to enable Oracle Social Network for multi-entity entity objects. This assumes the pre-condition of the accessor being as described at the beginning of this section.

- In Jdeveloper, open the definition of the Primary entity object class.
- Access the Java tab and click the Edit icon.
- Click Generate Entity Collection Class.
- Click OK to generate the EOCollImpl class for the entity object.
- Open the EOCollImpl class for the entity object in the source editor.
- Add the following code:

```java
@Override
protected EntityImpl add(EntityImpl srow)
{
    OAEntityImpl eo = (OAEntityImpl) super.add(srow);
    byte state = srow.getPostState();
    if (EntityImpl.STATUS_UNMODIFIED == state ||
        EntityImpl.STATUS_MODIFIED == state)
    {
        eo.registerOsnExternalListener("SecondaryEOAccessor");
    }
    return eo;
}
```

- Change the SecondaryEOAccessor to be the name of the accessor attribute in the Primary entity object that returns the OAEntityImpl (or sub-class) for the Secondary entity object.

**What happens when the secondary Entity Object Accessor Registration occurs**
The "add" method override accomplishes these actions:

- Ensures that Oracle Social Network integration on the primary entity object exists before performing any registration.
- Ensures that the accessor provided refers to an instance of an OAEntityImpl object.
- Checks to see that attributes in the Secondary entity object are to be sent to Oracle Social Network as marked in the view object. If there are no fields from the Secondary entity object to be sent, then no listener will be registered. If there are fields to be sent to Oracle Social Network, then an internal reference is set up in
the instance of the Secondary entity object marking that column it is to track for Oracle Social Network.

At runtime, each entity object will track which fields are to be sent to Oracle Social Network (if any). When a post action occurs, the Secondary Entity Object will notify the Primary Entity Object that it has been modified. The Primary Entity Object retrieves the associated view object to collate all fields that have been traced for sending to Oracle Social Network from all participating entity objects. The view object then will perform the send to Oracle Social Network. The Primary Entity Object will ensure that if multiple Secondary Entity Objects are modified, then only one send to Oracle Social Network will occur.

What happens if "add" in the Primary EOCollImpl is not performed and attributes from the Secondary entity object have been marked as Oracle Social Network enabled in the view object?

There are three types of events that will occur:

- Primary entity object fields only updated - The post event occurs on the Primary entity object and those fields in the Primary entity object will be sent to Oracle Social Network.

- Primary entity object and Secondary entity object fields updated - When the post event occurs, it will fire on the Primary entity object. Because the send is done from the view object, all fields that can be sent to Oracle Social Network from both the Primary and Secondary entity objects will be sent to Oracle Social Network.

- Secondary entity object fields only updated - Only a post transaction event will occur on the Secondary entity object and therefore no data will be sent to Oracle Social Network.

What if Multiple Updateable Entity Objects are included in View Object as non-reference

Having multiple entity objects included in the view object is generally not recommended by the BC4J framework but it is possible to implement. The complication for Oracle Social Network integration is that it uses the non-reference entity object to identify the Primary Entity Object. If there are multiple non-reference entity objects within the view object, then it is most likely that the Oracle Social Network integration will choose the incorrect entity object to be the Primary Entity Object. The solution to this is to add the fnd:OSN_PRIMARY_EO property to the view object’s properties and provide the EntityUsage name of the entity object to use as the Primary Entity Object. This property can only be added in the view object source because there intentionally is no Property Inspector fnd:OSN_PRIMARY_EO property.

This example demonstrates the use of the fnd:OSN_PRIMARY_EO property:

```xml
<ViewObject
 xmlns="http://xmlns.oracle.com/bc4j"
 Name="MeoDeptAttachNonReferenceVO"
 xmlns:fnd="http://xmlns.oracle.com/apps/entityview">
 <Properties>
  <SchemaBasedProperties>
   <fnd:OSN_PRIMARY_EO Value="MeoFndDemoDeptEO"/>
  </SchemaBasedProperties>
 </Properties>
 <EntityUsage
 Name="MeoFndDemoDeptEO"
 Entity="oracle.apps.fnd.applcore.socialnetwork.model.test.setup.multieo.attach.MeoFnd"
/>
</ViewObject>
```
By identifying which entity object is the Primary Entity Object for the view object, the Manage Oracle Social Network Objects UI will correctly configure the Oracle Social Network integration to ensure all send requests are sent using the declared Primary Entity Object.

**Note:**

`fnd:OSN_PRIMARY_EO` only needs to be implemented where there are multiple entity objects in the view objects that are flagged as non-reference. If there is only one entity object in the view object as non-reference, do not set this flag.

---

**When not to use multiple updateable entity objects Oracle Social Network Integration for a view object**

Consider this scenario:

- There are two view objects: the first view object (FirstVO) has two entity objects contained within it (OneEO + TwoEO) and the second view object (SecondVO) has only one entity object in it (TwoEO).
- The secondary entity object of the first view object (TwoEO) is the primary, and only, entity object of the second view object.

In this scenario, it does not make sense to use Oracle Social Network integration as there will be a synchronization issue. Since there are two view objects that contain data from TwoEO, there will be two Social Objects in Oracle Social Network that contain this data. When TwoEO data is updated from the SecondVO, only the Social Object that relates to the SecondVO will be updated. There is no way to determine that the FirstVO has been affected and its data should be re-synchronized to Oracle Social Network. The following is the recommended outcome to ensure that the data remains synchronized on Oracle Social Network:

- Do not perform the Secondary entity object registration steps as outlined in the Additional Advanced Configuration - View Objects That Have Multiple Updatable Entities section.
- Leave the second view object to send all updateable information relating to the secondary entity object (TwoEO).
- In the first view object, only include attributes from the secondary entity object that will not change after the Social Object has been created. For example: Social Object title or primary key value.
- In the first view object you should include attribute values that will help an end user find the associated Social Object for the second view object. For example: Social Object title or primary key value.

There is a future feature that has been mentioned (but has no schedule for implementation) where two social objects can be associated with each other. This feature, when implemented and uptaken by ATG-OSN integration, will allow the end user to quickly navigate between the two associated social network objects that will make the above recommendation more user friendly.
67.2.1.10 Configuring Multiple View Objects to Share the Same Social Object

A common scenario in Fusion Applications is to have two different UI pages for a single Business Object:

- A summary page that outlines the information in a logical read-only way and may include other information that is logical to show with that object.
- An edit page where the information about that object can be modified.

To achieve this in Fusion Applications, two different view objects are required (one per page) that refer to the same business object (such as Opportunity). Therefore, a logical business requirement would be to have both view objects update and view the exact same Social Object on the Oracle Social Network server. This would require Oracle Social Network to enable both view objects individually, because they need to be able to access the Oracle Social Network server independently of each other. The following instructions demonstrate how to share the same Social Object when there are two different view objects that can provide the attribute definitions, translations, and so on.

Before you begin

Pick one of your view objects which will be the Reference View Object. The Reference View Object:

- Is a view object that has already been Oracle Social Network-enabled using these instructions.
- Will be used as an "Oracle Social Network integration template" for all other view objects that will share the same Social Object.

In the Reference View Object, make a note of the following:

- Property values for these properties:
  - Nav Web Application
  - Nav View Id
  - Nav Page Params List
  - Nav Taskflow Id
  - Nav Task Key List
  - Nav Task Params List
  - Task Label
  - Task Label Attribute
- Names of all attributes that are Oracle Social Network-enabled.
- For each attribute, note the attribute property values for these properties:
  - Oracle Social Network Attribute Name Override
  - Oracle Social Network Gadget Type
  - Oracle Social Network User Key
67.2.1.10.1 Configuring the Reference View Object to Be Enabled by Oracle Social Network

To configure the Reference View Object to be enabled by Oracle Social Network:

- Open the Reference View Object and in the property inspector set the Applications > Oracle Social Network > Definition Name property. This will ensure that the view object's Oracle Social Network definition name will be consistent among all the view objects that are sharing the Social Object. This definition name must have the application short name at the beginning, and should be unique from all other existing view objects that are not to share the Social Object.

- [Optional] If there are any translated values that are different between the two view objects, set the Registration VO property on the view objects to true. This will set the fn:OSN_REGISTRATION_VO property on the view objects and ensure that any translations are read from this view object.

67.2.1.10.2 Configuring the Other View Objects to Be Enabled by Oracle Social Network

For each view object (not the Reference View Object):

1. Open the view object.

2. Set the Applications > Oracle Social Network > Definition Name to be the exact same value as the Reference View Object set in the previous section.

3. Set the property values to the values noted in the "Before you begin" steps.

4. Open the Add OSN View Instances wizard by right-clicking the Application Module containing the view objects and selecting the wizard.

5. For each attribute in the view object that matches an Oracle Social Network-enabled attribute in the Reference View Object:

- Set the Oracle Social Network enabled flag to True.

- Set the Oracle Social Network Gadget Type, Oracle Social Network User Key, Object Key and Currency Code Attribute property value to those noted in the Before you begin steps.

- Has the Oracle Social Network Attribute Name Override been set in the attribute of the Reference View Object?

  - If Yes, set the Oracle Social Network Attribute Name Override to be the same value between the two attributes.

  - If No, set the Oracle Social Network Attribute Name Override to be the name of the attribute in the Reference View Object unless the two attribute names are the same.

6. Open the Add OSN View Instances wizard by right-clicking the Application Module containing the view objects and selecting the wizard.

7. Shuttle the view objects from the Available View Objects to the Data Model.
This will ensure that all view objects are set up to use the same Oracle Social Network Social Object.

**Social Link Configuration**

Because each view object now is set up to match the Social Object definition in Oracle Social Network, the row from the view object that is on the page that is being displayed can be used as the osnRow in the applications panel.

### 67.2.1.11 Configuring Transient Entity Objects and View Objects

Under certain circumstances, there is a requirement to send data to Oracle Social Network without having an entity object that is based upon a database table. For example, a team could be using an Oracle Enterprise Scheduler job to update data that does use an entity object but still want that data sent to Oracle Social Network. The way to do this would be to use a Transient entity object and Transient view object.

Sending to Oracle Social Network from a Transient view object or entity object is possible with Oracle Social Network integration, however, the solution does have some restrictions. Performing a send to Oracle Social Network that creates a record is relatively straightforward. The entity object and view object were created using the wizard in JDeveloper and all attributes manually entered. The view object was then added to an application module that will control the transaction in which this will occur. A transaction is required for this functionality because the send to Oracle Social Network happens upon the commit of the transaction. The configuration of the Oracle Social Network properties and setup UI is identical to a regular view object, except there must be a minor fix to the VOImpl to change its default of looking at database tables, to directly looking at the entity cache so that the Oracle Social Network integration can find the row upon commit. This is done using the following code:

```java
@override
protected void create()
{
    super.create();
    setQueryMode(QUERY_MODE_SCAN_ENTITY_ROWS | QUERY_MODE_SCAN_UNPOSTED_ENTITY_ROWS | QUERY_MODE_SCAN_VIEW_ROWS);
}
```

**What happens when a transient entity object/view object is used for Oracle Social Network integration**

When a row is created or inserted in the transient view object, populated with attribute data, and the commit happens, the row’s data is sent to Oracle Social Network as a create request. This works well for transient entity objects and view objects.

It is during an Update that the situation becomes less clear. Due to the nature of transient view objects and entity objects, such as when they have no persistent state, it is impossible to change their entity state away from STATUS_NEW/INITIALIZED until a commit occurs on that instance of the transient entity object or view object. The Oracle Social Network integration uses the entity state to determine whether to perform a create/update/delete request to Oracle Social Network. However, the Oracle Social Network client library handles the update issue. When a second create request is sent to Oracle Social Network that contains all of the Social Object fields (view object attributes) that were sent to Oracle Social Network originally, it will detect that the Social Object was already created and only will perform an update of the fields whose values have changed between the two Oracle Social Network sends. The only issue with this is in the detail: *all* fields need to be sent the second time, even if the field value has not changed. If a field value is not supplied, it will be treated as if
null was sent and the field value will be removed from Oracle Social Network, but if the field value is supplied and its value has not changed, then the field will be treated as not-updated.

Therefore, the update solution becomes a non-issue as the transient view object can be populated in the exact same mechanism and committed in the same way as the original insert. When the payload of the second create request gets to Oracle Social Network, it will determine the correct course of action for the insert/update of the Social Object data.

**67.2.1.12 Working with Special Cases**

This section discusses situations that are not covered elsewhere.

- **Date Effective Entity Objects**
  A date effective entity object is one in which an update on an existing entity object row puts an end date on it and creates or inserts a new row with the updated details. In Oracle Social Network, this should only submit the details of the update and not the insert of the new row. Only the last chronological date effective row data will be sent to Oracle Social Network. This is the date effective row where the Effective End Date is 31-Dec-4712. Data from any other date effective row that does not contain that end date will not be sent to Oracle Social Network.

  It is possible for the last chronological date effective row to be effective from a future date. To communicate this to the viewer, a new attribute "As Of" automatically will be included for a date effective object when it is enabled for Oracle Social Network. The value of this attribute will come from the attribute that contains the start date of the date effective range.

- **Parent-Child View Objects**
  You can send view object rows for both the parent and child rows individually, but you cannot send both Social Objects together as a Master - Detail since Oracle Social Network does not support this relationship. There are ways to view the parent from the details rows using the parent id and tie them together through URLs in the Oracle Social Network UI.

- **Descriptive Flexfields**
  The Oracle Social Network integration supports descriptive flexfields which have been deployed for an Oracle Social Network-enabled view object. An Oracle Fusion applications developer does not need to configure any properties at design time to enable the descriptive flexfield integration. When the Functional Administrator uses the "Manage Social Network Objects" UI to add attributes to send to Oracle Social Network, the descriptive flexfields attributes will also be shown in the list in a tree structure organized by value set.

**67.3 Implementing the Oracle Social Network Setup UI**

There are several features of the Oracle Fusion Applications-Oracle Social Network integration that can be used when designing Oracle Fusion applications user interfaces and their application.

**67.3.1 Configuring the Setup Application**

The task flow shown here will be available for users who will configure the Oracle Social Network integration on a client site.
The "Manage Oracle Social Network Objects" task flow definition and its available parameters are:

- **Component**: Oracle Social Network
- **Task flow**: Manage Oracle Social Network Objects
- **Function Security Privilege**: FND_MANAGE_SOCIAL_NETWORK_TRACKING_PRIV
  
  **Display Name**: Manage Oracle Social Network Tracking

- **Task Flow Invocation Artifact**:

  /WEB-INF/oracle/apps/fnd/applcore/socialnetwork/ui/flow/SocialNetworkTF.xml#SocialNetwork

- **Parameters**
  
  - `[moduleType] Note: If moduleType is passed, moduleKey also must be passed. Together they form a unique key for a module.
  
  - `[moduleKey] Note: If moduleKey is passed, moduleType also must be passed. Together they form a unique key for a module.
  
  - `[pageTitle]`

- **Description**: Manage the Social Network Tracking setup.

The parameter values that can be provided are shown in Table 67-4.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>moduleType</td>
<td>The type of module to filter the list of application modules shown. This parameter is used with the moduleKey to perform the filtering. The application modules contain the view objects that are available to be sent to Oracle Social Network. Examples: APPLICATION, FAMILY</td>
<td>When no value is supplied, all modules will be shown.</td>
</tr>
<tr>
<td>moduleKey</td>
<td>The module key to filter the view objects in the task flow. The mapping between the module key and the application modules comes from the fnd-osn-integration.xml setup in the &quot;Add OSN View Instances&quot; wizard.</td>
<td>When no value is supplied, all modules will be shown.</td>
</tr>
<tr>
<td>pageTitle</td>
<td>The title to show at the top of the &quot;Manage Oracle Social Network Objects&quot; page.</td>
<td>&quot;Manage Oracle Social Network Objects&quot;</td>
</tr>
</tbody>
</table>

In Functional Setup Manager, Applications Core will add the "Manage Oracle Social Network Objects" task to provide access to the setup task flow with no moduleType or moduleKey specified. This task will give access to the task flow for managing any Oracle Social Network object in Oracle Fusion Applications.

You can create your own product module-specific Oracle Social Network setup tasks in your own task lists that provide access to the setup task flow but that are filtered by the relevant modules. For example, Financials can create a Manage Financials Oracle Social Network Objects by passing in the correct moduleType and moduleKey.
If a valid `moduleType` and `moduleKey` pair is specified, the UI provides access to all objects whose modules roll up to the specified module. That is, if a family is specified, the UI displays all objects whose modules roll up to the family.

**Note:**
Oracle Social Network Object settings are moved using the customization sets on the Customization Migration page instead of using the export and import function in the Setup and Maintenance work area of Functional Setup Manager. Customization Migration is not module-specific.

### 67.3.2 Accessing the Credential Store

As part of the Provisioning role, credentials must be created that allow the Oracle Social Network client libraries (the interface between Applications Core and Oracle Social Network) to create an authenticated connection to the Oracle Social Network server. To access the credential store, the Oracle Social Network library will need to be granted permission by the security of the containing application. The following code snippet must be added to the `jazn-data.xml` file so the Oracle Social Network client library can access the credential store:

```xml
<grant>
  <grantee>
    <codesource>
      <url>file:${domain.home}/servers/${weblogic.Name}/tmp/_WL_user/oracle.social.client/-</url>
    </codesource>
    <permission>
      <class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
      <name>context=SYSTEM,mapName=oracle.social.security,keyName=FUSION_APPS_OSN_INTEG_APP_ID-KEY</name>
      <actions>read</actions>
    </permission>
  </grantee>
  <permissions>
    <permission>
      <grantee>
        <codesource>
        </codesource>
        <grantee>
        </grantee>
        <permissions>
        </permissions>
      </permission>
  </permissions>
</grant>
```

Before adding this block to the `jazn-data.xml` file, verify if there is already a grant block for the URL `${domain.home}/servers/${weblogic.Name}/tmp/_WL_user/oracle.social.client/-`. If this already exists, then just add the `<permission>` node to the existing grant.

### 67.3.3 Avoiding Overwriting Registrations and Data

When there are multiple developers working on Oracle Social Network integration with the same view object, there is a high risk of the developers overwriting other developers' Social Object registrations and data on the Oracle Social Network server. To avoid this scenario, use the `oracle.applicore.socialnetwork.registrationprefix` system property. This property can be set on the Java Options command line to a different value for each developer and will be prefixed to the Oracle Social Network registration definition name. This will allow the same view object to be registered multiple times on the Oracle Social Network server, thus allowing each developer to have his or her own Social Object registration.
67.4 Adding Oracle Social Network to Your Application UI

Oracle Social Network provides an ADF declarative component to launch the Oracle Social Network UI within an Oracle Fusion application. This declarative component is rendered as a Social link and is available in two areas—Global Area and application panel.

Note that once the Social link is implemented, the application’s end users can see the Social link only if they are internal to the organization. That is, if MegaCorp is the organization running the Oracle Fusion application’s instance, MegaCorp’s internal employees can access the Social link, but the company’s external suppliers and partners cannot. Access to the Social link in the two aforementioned locations is controlled by the Launch Oracle Social Network privilege that is granted to internal users by the Worker duty role.

When the Social link is clicked, it launches the Oracle Social Network UI that is displayed in a layover panel on the right side of the Oracle Fusion application page.

The ADF declarative component is available in four versions for use in various areas of the Oracle Fusion applications pages:

- image +link
- link only
- image +button
- button only

The component itself has been integrated out of the box as a link in two places using Applications Core and is available for use by developers.

The Social link has been added as a text-only link to the Global Area/Menu as shown in Figure 67-2. Therefore, the link will be available on all Oracle Fusion application pages.

Figure 67-2 Social Link in the Global Area

Note:

Language and time zone preference settings in Oracle Fusion Applications and Oracle Social Network are not integrated. Preferences set in Oracle Fusion Applications do not propagate to the Oracle Social Network server.

When the Social link is clicked, the Oracle Social Network UI is launched on the right side as shown in Figure 67-3. This view displays all the Conversations, Social Objects,
Users and Groups to which a user has access (displayed within the All tab in the Oracle Social Network UI).

**Figure 67-3  Example of the Oracle Social Network Popup**

The Social link also is added to the left of the toolbar (buttons) in the application panel. The link will appear automatically in Oracle Fusion application pages if the application panel has been used for object detail pages.

Clicking the **Social** link in the application panel also launches the Oracle Social Network UI on the right side as a layover panel. In this view, Oracle Social Network UI is displayed in the context of the current Oracle Fusion application page or object.

The Social link in the Application Panel typically appears as shown in **Figure 67-4**.

**Figure 67-4  Example Showing the Social Link in an Applications Panel**

The link currently is supported only within the application panel. The Social link should not be directly consumed outside an application panel. Also, you should not use the Oracle Social Network ADF declarative component (link/button) directly.

When the Social link is clicked from within the application panel, the Oracle Social Network UI should appear as shown in **Figure 67-5**. Within the Oracle Social Network UI there are two tabs:

- In the context of the Oracle Fusion application Object (depending on the context passed as explained below), the Oracle Social Network UI displays the associated Social Object and any Related Conversations.
• The All tab displays all the Conversations, Social Objects, Users and Groups to which a user has access.

**Figure 67-5  Example Showing Popup from Applications Panel**

67.4.1 How to Render the Behavior of the Social Link in an Application Panel

There are some checks made internally within the application panel and in the Oracle Social Network ADF declarative component that control the display of the Social link in the Oracle Fusion application pages.

The Social link in the application panel will render only when some context is passed to the application panel. The rules that control the rendering are:

• If the Oracle Social Network URL is not configured, the link will not show (it also will not show in the Global Menu).

• If there is no context passed to the application panel, the link will not render. The end-user will need to use the link from the Global Menu to see all Oracle Social Network Conversations and Social Objects.

• If the view object is not enabled (out of the box using Design Time) for Oracle Social Network, then the link will not render. See Configuring the Business Object for Oracle Social Network Integration.

• If the view object is enabled (out of the box using Design Time) for Oracle Social Network, but the Oracle Social Network task flow has not been enabled using Oracle Fusion Functional Setup Manager, then the link will not render.

**Note:**

Developers can control the rendering of the Social link by setting an Expression Language expression on the `osnRendered` attribute (see Table 67-5). For example, when the same UI is used for both create and edit cases, the developer may want to hide the Social link in the Create case because the business object may not be ready at that point.

67.4.2 How to Pass Context and Parameters to an Application Panel

This step must be performed for the objects (context) that are enabled for Oracle Social Network.

The application panel exposes a subset of parameters that can be used to control the behavior of the Oracle Social Network Social link. Some additional parameters
exposed by the Oracle Social Network Social ADF component are used internally by
the application panel and are not exposed. Table 67-5 lists the parameters available to
control the behavior of the Social link and pass the context to it:

Table 67-5  Parameters that Control the Behavior of the Social Link

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Type</th>
<th>Description</th>
<th>Valid Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>osnDisable</td>
<td>Boolean</td>
<td>Explicitly controls whether the Social link (if rendered) is disabled or not. This is optional.</td>
<td>true or false. The default is false.</td>
</tr>
<tr>
<td>osnRendered</td>
<td>Boolean</td>
<td>Explicitly controls (apart from internal checks made) whether or not to render the Social link. This is optional.</td>
<td>true or false. The default is true.</td>
</tr>
<tr>
<td>osnRow</td>
<td>oracle.jbo.Row</td>
<td>An Expression Language expression that returns a view object Row that must contain the Social Object ID (that is, the primary external ID).</td>
<td></td>
</tr>
<tr>
<td>osnType</td>
<td>String</td>
<td>The fully-qualified view object type/ID as registered with Oracle Social Network through Functional Setup Manager&quot;Manage OSN task flow.&quot;</td>
<td>Do not use.</td>
</tr>
<tr>
<td>osnTypeGroup</td>
<td>String</td>
<td>The application module ID that was used to group together a set of view objects for Oracle Social Network.</td>
<td>Do not use.</td>
</tr>
<tr>
<td>osnDisableShare</td>
<td>Boolean</td>
<td>Controls whether the Oracle Social Network UI should show the Share button when the current row is not yet shared or sent to Oracle Social Network. This parameter is only useful when osnRow is passed.</td>
<td>true or false. The default is false.</td>
</tr>
<tr>
<td>osnDisableJoin</td>
<td>Boolean</td>
<td>Controls whether the Oracle Social Network UI should allow the current user to Join (become a member of) the Social Object (if shared). This parameter is only useful when osnRow is passed.</td>
<td>true or false. The default is false.</td>
</tr>
</tbody>
</table>

Examples of Parameters

- For a specific object/row (only osnRow is specified)

  <fnd:applicationsPanel Id="ap1"
    osnRow="#{bindings.Dept1.currentRow}"
  .../>

- For a specific object/row ID but disabling Share

  <fnd:applicationsPanel Id="ap1"
    osnRow="#{bindings.Dept1.currentRow}"
    osnDisableShare="true"/>

67.4.3 How Share and Join Behave

The Share and Join buttons are displayed within the Oracle Social Network UI that is
launched from the Social link in the application panel. These actions are applicable
only in the contextual view displayed for an Oracle Fusion application object.

The core state transitions for the Share and Join buttons are:
• The **Share** button is displayed when the view object is enabled with Manual mode and the object has not yet been shared in Oracle Social Network. In this case, the **Share** button will be displayed and the user can choose to create the Oracle Social Network Social Object manually. In Automatic mode, you should not see the **Share** button unless the business object was created before it was Enabled for OSN. This is because the Oracle Social Network Social Object will be created automatically when the Oracle Fusion application business object is created.

• Once a user clicks the **Share** button, the button will not be displayed again, unless there is an error in sharing with Oracle Social Network. The Oracle Social Network UI then displays the Social Object.

• The **Join** button is displayed for both Manual and Automatic modes for the current user who is not yet a member of the Social Object in Oracle Social Network. The **Join** button will be displayed only after the object is shared, either automatically or manually by another user, in Oracle Social Network.

• Once a user clicks the **Join** button, the user is made a member of the Social Object in Oracle Social Network. The Oracle Social Network UI then displays the Social Object and any related conversations.

• The **Join** and **Share** buttons can be hidden by setting the `osnDisableShare` and `osnDisableJoin` parameters to the application panel. See Table 67-5.

### 67.5 What Happens When Functional Administrators Manage Oracle Social Network Objects

The Functional Administrator will use the Functional Setup Managers supplied by the product team to take the set of view objects available for Oracle Social Network integration and choose which are to be enabled for sending data to the site's Oracle Social Network instance.

**Prerequisites**

- The deployed JAR files containing the view object definitions and the `META-INF/fnd-osn-integration.xml` file must be in the classpath of the product team's Functional Setup Manager.
- The Functional Setup Manager has had the "Manage Social Network Objects" task flow added to it.
- The person fulfilling the provisioner role has set up the Oracle Social Network location and the Oracle Social Network credential.

#### 67.5.1 Enabling a View Object for Oracle Social Network Integration

When the functional administrator loads the Functional Setup Manager, a list of tasks is displayed. One will be "Manage Social Network Objects." After this task is selected, the setup UI for enabling the view objects for Oracle Social Network integration will be shown. When the "Manage Social Network Objects" page has finished loading, the list of view objects that the developer made available for Oracle Social Network can be seen.
67.5.2 Change the Oracle Social Network Enabled Status

To enable or disable the view object for Oracle Social Network integration at the site when the "Manage Social Network Objects" page is loaded, the configurer can use the following steps:

1. Select the view object in the Business Objects tree that is to be Oracle Social Network enabled.

2. Click Enable OSN Tracking.

3. When the popup window is displayed, choose either:
   - Manual - The end user consuming the Oracle Fusion application will need to click the Share button to send the view object row to Oracle Social Network.
   - Automatic - When the end user commits the view object row with which they are working, the send to Oracle Social Network will occur automatically.
   - None - Disable Oracle Social Network integration.

4. Click OK.

The view object shown in the tree now will have the updated Enabled status to match the selection. There is a shortcut to disabling Oracle Social Network integration for a view object by selecting it in the Business Objects tree and clicking Disable OSN Tracking.

67.5.2.1 How to Add and Remove Attributes to Send to Oracle Social Network

Changing the OSN Enabled state of the view object is only the first half of the configuration that is required to send a Social Object to Oracle Social Network. The attribute values that will become the Social Object fields also need to be selected. Use these steps to select the attributes that will be Oracle Social Network enabled:

1. Select the view object in the Business Objects tree that is to have attributes Oracle Social Network enabled. This should show the list of Oracle Social Network enabled attributes in the table at the bottom of the page.

2. Click New. This will display a dialog showing all of the attributes that the developer marked as Oracle Social Network integration available.

3. Check or uncheck the check boxes next to each attribute to add and remove attributes. Attributes that are selected are Oracle Social Network enabled.

4. Click OK.

The attributes that were checked in the dialog will now be shown in the table at the bottom of the page. This is the list of attributes for the view object that have been marked as Oracle Social Network enabled. There is a shortcut for marking the attributes as not Oracle Social Network enabled. Select the attribute and click Delete to remove it from the list.

67.5.3 Committing the Oracle Social Network Enabled View Objects

Until this point, there has been no change to the stored configuration of the Oracle Social Network Enabled view objects. This means that if the functional administrator leaves this page or cancels, then the way in which the Oracle Fusion application
integrates with Oracle Social Network at runtime will not have changed. By clicking Save at the top of the page, the information about all displayed view objects' Oracle Social Network enabled status and their attributes are committed locally for use at runtime and the definition of the Social Object based upon the selections made by the developer for the view object will be built and sent to Oracle Social Network.

What happens when Save is clicked

Two key functions happen when Save is clicked:

- **Oracle Social Network Enabled Configuration Is Saved Locally**
  
  When the developer saved the Oracle Social Network configuration for the view object, all of that information was stored as properties in the view object XML definition file. A very similar process happens for enabling the view object for XML. When Save is clicked, the custom properties for these view objects are changed:

  - `fnd:OSN_VO_ENABLED_BY_CUSTOMER` - Stores whether the user selected Automatic/Manual/None for the Enabled status.
  
  - `fnd:OSN_ENABLED_ATTR_BY_CUSTOMER` - Will be set to "true" on all attributes that the user selected to be Oracle Social Network enabled.

  In addition to the view object property being modified, the primary entity object for the view object will have these custom properties set upon it:

  - `fnd:OSN_EO_ENABLED_BY_CUSTOMER` - A mirror of the view object's `fnd:OSN_ENABLED_ATTR_BY_CUSTOMER` property.
  
  - `fnd:OSN_VIEW_OBJECT_NAME` - The fully-qualified name of the view object that will need to be used to send data to Oracle Social Network when a Data Manipulation Language (DML) event occurs on this entity object.

  Since the view object and entity object definition XML cannot be changed at runtime, these modifications will be stored in Oracle Metadata Services (MDS) at the persdef layer. For each view object that was displayed in the "Manage Social Network Objects" there will be four persdef files and entries stored. An example of these files are for the SndDemoDeptVO:

  ```
  persdef/oracle/apps/model/entity/mdssys/cust/site/site/SndDemoDeptEO.xml.xml
  persdef/oracle/apps/model/entity/SndDemoDeptEO.xml
  persdef/oracle/apps/model/view/SndDemoDeptVO.xml
  persdef/oracle/apps/model/view/mdssys/cust/site/site/SndDemoDeptVO.xml.xml
  ```

  These entries can be imported and exported if the enabled Oracle Social Network integration configuration must be ported between server instances.

- **Social Object Definition Is Sent to Oracle Social Network**

  After the view object Oracle Social Network enabled configuration has been stored into MDS view objects, the Oracle Social Network structure must be sent to Oracle Social Network as a Social Object definition. All view objects that have an Enabled status of Manual or Automatic will have the definition sent to Oracle Social Network. The Social Object definition provides details such as the attributes (name, data type) as Social Object field definitions and the translations for all of the attributes to be used as the field labels when a Social Object is shown to the user in Oracle Social Network.

  To see the registered definition on Oracle Social Network and confirm that it was correctly sent, the Oracle Social Network social definition can be directly retrieved
using the REST API. To return the Social Object definition, create a request in the following structure from within a web browser: `<Oracle Social Network URL>/social/api/v1/socialDefinition/<definition name>

The definition name is either the value of the fnd:OSN_DEFINITION_NAME property set on the view object by the developer or the family module name concatenated with the view object name (if the definition name was not set).

Using the SndDemoDeptVO example again, if it is in the FND family with an Oracle Social Network server of http://server_name.company.com:9003/osn, it would resolve to the Social Object definition URL of: http://server_name.company.com:9003/osn/social/api/v1/socialDefinition/FND.oracle.apps.model.view.SndDemoDeptVO.

When Save is clicked, a progress popup will be shown that displays the current status of the save and send to the OSN server.

67.5.4 Updating Translations

"Manage Social Network Objects" also provides updated translations. When the Social Object definition has been sent to Oracle Social Network by clicking Save, the initial set of translations is sent to the Oracle Social Network server for the view objects. However, users on a site can choose to enable more languages within Oracle Fusion Applications by loading language bundles. The Update Translations button provides a quick way to send all the Oracle Social Network enabled view object translations for all enabled languages to Oracle Social Network. For example, a site has been running in English and has registered the view objects as Social Object definitions on Oracle Social Network. The site then loads the French language. The site then can use Update Translations to send both the English and French translations for the Oracle Social Network enabled view object to Oracle Social Network, thereby allowing the user to see all of the field labels in French.

An individual view object can have its translations updated in isolation by selecting a view object in the Business Objects table and clicking Update Translations at the top of the Business Objects table.

What happens when clicking Update Translations

Each view object that is displayed that has an Enabled state of Automatic or Manual will have its translated strings for the view object and its attributes loaded and sent as a social definition update to Oracle Social Network. The only values that can change in this upload are the translated strings. To confirm if the new translations are correctly seeded in Oracle Social Network, the social definition URL for the view object can be used to see the translated strings for the Social Object definition.

When the page-level Update Translations button is clicked, a progress popup will be shown that displays the current status of the save and send to the Oracle Social Network server. The progress indicator is not shown for the Update Translation button in the Business Objects table.

67.5.5 Synchronizing

If there is an issue with a previous send to the Oracle Social Network server within the "Manage Oracle Social Network Objects" UI, or if a migration of Oracle Social Network integration configuration has occurred between servers, then the state of the definition on the Oracle Social Network server will be different from what is stored in persdef. To correct this, the Synchronize button was added. This will take the definitions of all view objects and resend them to Oracle Social Network. An individual view object can
be synchronized in isolation by selecting a view object in the Business Objects table and clicking Synchronize at the top of the Business Objects table.

**What Happens When Clicking Synchronize**

Each view object that is displayed and that has an Enabled state of Automatic/ Manual, will have its Oracle Social Network definition re-created and sent to the Oracle Social Network server. To confirm if the synchronized view objects are correctly seeded in Oracle Social Network, the social definition URL for the view object can be used to verify the Social Object definition. When the page-level Synchronize button is clicked, a progress popup will be shown that displays the current status of the send to the Oracle Social Network server. The progress indicator is not shown for the Synchronize button in the Business Objects table.

### 67.6 What Happens When End Users Interact with Oracle Social Network Object Instances

The only decision an end user needs to make is to share instances when Manual integration mode is enabled.

**Automatic Integration**

When Oracle Social Network is enabled as Automatic for a view object, end users will not see any visual signs that a send to Oracle Social Network has occurred with the data they just committed. The first that they will know about this is when they check in Oracle Social Network and the record has been created in Oracle Social Network. Any future updates or deletes (Oracle Social Network Conversation Close) also will be reflected in Oracle Social Network upon any further commits of the row.

**Manual Integration**

When Oracle Social Network is enabled as Manual for a view object, end users will be able to see the Oracle Social Network Share button on the page. If the users create or update the record, they will not see any information about that view object row in Oracle Social Network. After they have committed the row, they can click Share, which will do the initial send of the row data to Oracle Social Network. By clicking Share, after the initial record has been sent to Oracle Social Network, a new window will appear with the Conversation displayed within it (if the socialShare component was set to launch=true when it was configured in the page). Any future updates or deletes (Oracle Social Network Conversation Close) also will be reflected in Oracle Social Network upon any further commits of the row.

### 67.7 Test End-To-End Flow

The end-to-end test flow provides a basis for testing the Oracle Social Network integration implementation points to prove the success of the integration.

Testing requires that the roles for Oracle Fusion Developer and Provisioner have been completed successfully.

Follow these steps to test the flow:

1. After configuring the Oracle Fusion Developer role, deploy the model library that contains the META-INF/fnd-osn-integration.xml file and the view object definitions into an ADF Library JAR file.
2. Include this JAR file as a library in the Functional Setup Manager project.
3. Deploy and run the Functional Setup Manager in an Integrated WebLogic Server environment.

4. Log into the Functional Setup Manager and choose the Manage Social Network Objects task.

5. Find the view object that has been configured as available for Oracle Social Network integration, change it to have an Enabled state of **Automatic** and add all the attributes to the list of Oracle Social Network enabled attributes.

6. Click **Save** to keep the Automatic enabled state and attribute configuration, and send the Social Object registration to Oracle Social Network.

7. Deploy and run the application that uses the deployed library from the first step.

8. Log in to the application and find the product page that uses the view object.

9. Create a new row for the view object, fill in the required data and commit the row.

10. Log in to Oracle Social Network using the same user as logged in to the application and see the Social Object representing the newly-created or committed view object row.

---

**Note:**

The same MDS partition **must** be used for both Functional Setup Manager and for Oracle Fusion Applications. If this is not done, the Oracle Social Network integration will not occur between the two applications.
Work with the Application Taxonomy

This appendix describes the theory of the Oracle Fusion application taxonomy, how to view the taxonomy, and how to extract taxonomy data from a table and how to insert taxonomy data into a table.

This appendix describes:

• The theory of the application taxonomy
• How to view the taxonomy
• How to extract taxonomy data from a table and how to insert taxonomy data into a table

This appendix includes the following sections:

• Introduction to the Oracle Fusion Application Taxonomy
• Working with Objects and Methods in the Application Taxonomy
• Understanding Taxonomy MBeans

It is important to note that there is no tool for working with the taxonomy; developers use public business objects and do all work within JDeveloper. In general, only developers who are referring to modules, such as Application, will need to work with the taxonomy.

68.1 Introduction to the Oracle Fusion Application Taxonomy

The Oracle Fusion application taxonomy organizes the artifacts and functions of Oracle Fusion Applications into a hierarchical structure. Every Oracle Fusion development artifact or file is tagged. The structure starts with the Application Line and extends through the Logical Business Area.

Application Line
  Application Family
    Application
      Logical Business Area
      Logical Business Area

In the taxonomy user interface, the hierarchy would appear similar to the example shown in Figure 68-1:
68.1.1 Characteristics of the Level Categories

The taxonomy hierarchy provides a map of the dependencies that exist within an application and across applications.

Seed Data

The Oracle Fusion Applications Design Repository (ADR) team has provided Taxonomy seed data for the following levels in the hierarchy: Application Line, Family, Application, and Logical Business Area (LBA).

You can create as fine-grained an application taxonomy as you wish. You can break up an application into sub-applications or pseudo applications. For example, there are many setup use cases where an overall process is made up of many smaller subordinate processes.

68.1.2 How to Manage the Lifecycle

The applications taxonomy is especially useful in managing various phases of the application lifecycle. These phases include:

- Installation and Deployment
- Patch Creation
- System Administration
- Diagnostics and Maintenance

68.1.2.1 Creating Patches and Patch Sets

Patches and patch sets can be constructed based on the data defined in the taxonomy. You can choose any node of the taxonomy as the source for a patch file manifest. That starting node can scale all the way up to the top of the application taxonomy tree for a new release for the entire suite.

68.1.2.2 System Administration

System administrators monitoring performance, processes, system use, and so on, can use the application taxonomy to organize information and navigate to the level of detail they require. Administrator dashboards will start at higher levels of the taxonomy to provide broad overview of system status. When trouble is detected, the taxonomy can be used to drill down to where attention is required.
Patches will be tagged with the versions they contain. When a patch is applied, dependency information in the taxonomy can be used to determine which parts of the system will be affected. This can be used to assess system testing requirements after the patch is applied, or to schedule partial downtimes while patching is in progress.

68.1.2.3 Diagnostics and Maintenance

Diagnostic tests, logging, error messages, online help, support bulletins, and other artifacts are tagged with the module and version in the taxonomy to which they pertain. When trouble is detected, information from the customer’s system can be matched with these tags to direct them to appropriate assistance.

If the problem cannot be resolved through diagnostics and help, the taxonomy can be used to search for patches available for a particular module and version. Patches could be available at any level of the hierarchy, from one-offs, through larger roll-ups. The taxonomy can be used to follow the troublesome module up through the hierarchy to search for larger roll-ups that might be relevant.

If support is required, the taxonomy can be used to automatically construct Support Information Bundles, containing version information, with the results of diagnostics registered for these components.

68.1.3 Benefits of a Logical Hierarchy

The organization of the application taxonomy does not need to match the physical file directory structure, which is unlikely to consistently correspond to the functionality provided by those files. File directory structures often serve to group files according to a high level file type (such as all seed data files in one directory, all Java files in another directory, and so on), rather than by their functionality.

The applications’ Java EE package structure is a simple physical hierarchy based on the directory structure into which you organize your runtime files on disk. It is identical to the package structure that you use when defining Java class file packages. The concept has been expanded to also support metadata files, such as JSPX files. However, the information maintained by the application taxonomy supports many functional capabilities that cannot be supported by the standard Java EE package hierarchy.

Many of the artifacts that comprise a given application are shared among various applications and modules. The relationships among applications and artifacts constitute a network rather than a simple hierarchy, and are essential when interrogating and modeling dependencies. There is no support for such an integrated map of relationships in Java EE package structures.

There are other critical business requirements that cannot be satisfied by using a physical directory structure to organize an application hierarchy. Customers will often extend or subclass various runtime components to customize the application behavior to meet their specific business needs. Over time, application teams will wish to refine or refactor their application hierarchies as they add more features and functionality. Teams will want to refine or reorganize modules that leverage various artifacts. The application taxonomy’s logical definition of applications and their related runtime components saves customers from having to modify their references to these packages if the logical hierarchy is changed.

68.1.4 Delivery Hierarchy

In the application taxonomy, the delivery hierarchy is the master source for all the directories and files that comprise an application.
The delivery hierarchy represents the relationships between files and the application team that is responsible for the development, maintenance, and delivery of those files. Nodes within the delivery hierarchy have unique parents, so there is one path through the delivery hierarchy to any given file.

### 68.1.5 How to Integrate Taxonomy Task Flows into Oracle Fusion Functional Setup Manager

Every application registers task flows with the Functional Setup Manager that provides a single, unified user interface that allows customers and implementers to configure all Oracle applications by defining custom configuration templates or tasks based on their business needs.

The Functional Setup Manager UI enables customers and implementers to select the business processes or applications that they want to implement. For example, a Human Resources application can register setup activities like "Create Employees" and "Manage Employee Tree Structure" with the Functional Setup Manager. Trees task flows then provide the mechanism for an application team to register an activity such as "Manage Employee Tree Structure," which in this case, is a tree structure task flow with the tree structure code parameter set to some HR tree structure. Table 68-1 lists the task flow and its parameters.

<table>
<thead>
<tr>
<th>Task Flow Name</th>
<th>Task Flow XML</th>
<th>Patterns Passed</th>
<th>Behavior</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manage Taxonomy Hierarchy</td>
<td>/WEB-INF/oracle/apps/fnd/applcore/taxonomy/ui/taskflow/ViewDeliveryHierarchy.xm#ViewDeliveryHierarchy</td>
<td>[pageTitle]</td>
<td>The Manage Taxonomy Hierarchy accepts the optional parameter [pageTitle] and navigates to the Taxonomy Delivery Hierarchy page.</td>
<td>This page serves as the starting point from which a user can select a particular node and perform the available actions, such as create, update, and view components for a selected node. From this page, a user can navigate to other task flows, such as Search Hierarchy, View Components and Search Components.</td>
</tr>
</tbody>
</table>

### 68.2 Working with Objects and Methods in the Application Taxonomy

You can use the application taxonomy at a lower level by using the public entity objects and view objects.

For example, you can create an association between the application team entity object, which has a foreign key reference to alternative_id in ApplTaxonomyPEO, and provided service methods to either join to the taxonomy table or to traverse through the taxonomy hierarchy using an API (for instance, to which Family does a given
Application belong?) or for other lookup information about the nodes (for instance, what is the short name for a given application?).

### 68.2.1 Particular Table Columns and Data

These items are applicable to some Oracle Fusion Middleware Extensions for Applications (Applications Core) tables.

**Who Columns**

All tables containing seeded or transaction data must include the Who columns shown in Table 68-2:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Null?</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATED_BY</td>
<td>VARCHAR2 (64)</td>
<td>Not Null</td>
</tr>
<tr>
<td>CREATION_DATE</td>
<td>TIMESTAMP</td>
<td>Not Null</td>
</tr>
<tr>
<td>LAST_UPDATED_BY</td>
<td>VARCHAR2 (64)</td>
<td>Not Null</td>
</tr>
<tr>
<td>LAST_UPDATE_DATE</td>
<td>TIMESTAMP</td>
<td>Not Null</td>
</tr>
<tr>
<td>LAST_UPDATE_LOGIN</td>
<td>VARCHAR2 (32)</td>
<td></td>
</tr>
</tbody>
</table>

If the table has "extended Who" columns used to track updates by Oracle Enterprise Scheduler Service programs, the columns must be changed to those shown in Table 68-3. You do not need to add extended Who columns if the table does not already have them.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Null?</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST_ID</td>
<td>NUMBER (20)</td>
<td>NULL</td>
</tr>
<tr>
<td>JOB_DEFINITION_NAME</td>
<td>VARCHAR2 (100)</td>
<td>NULL</td>
</tr>
<tr>
<td>JOB_DEFINITION_PACKAGE</td>
<td>VARCHAR2 (900)</td>
<td>NULL</td>
</tr>
</tbody>
</table>

**Replace RAW Columns with VARCHAR2**

Raw columns may only be used in internal tables that are never directly exposed in Oracle Application Development Framework (Oracle ADF).

- Columns containing user GUIDs should be changed to varchar(64).
- Columns containing all other GUIDs should be varchar2(32), and use rawtohex() to convert the raw GUID to a hex value.
- Raw columns containing anything else should be replaced with Binary Large Object (BLOB).
68.2.2 Denormalized Taxonomy Table

The ApplicationLine column of the Taxonomy table has been denormalized to allow data for multiple application lines to prevent hierarchical queries against the taxonomy table.

By default, the ApplicationLineCriteria will be applied on the view objects exposed on the view object by taxonomy (public and private) with the value of 1 for the Oracle Fusion application line. If you need to get data for another application line, you can set the appropriate value for the bind variable bProductLine.

The service methods use the default Oracle Fusion application line value of 1. The application module APIs that do not accept an application line id assume that the data is being queried for the Oracle Fusion application line.

If you query directly against the taxonomy tables, you must take into account the application line denormalization. You will have to add the filter product_line = <appropriate application line id> to prevent returning multiple rows for a given module key or id.

68.2.3 Available Public Business Objects

The following public entity objects are located in the oracle.apps.fnd.applcore.model.publicEntity package and are exposed by Applications Core:

- ApplTaxonomyPEO
- ApplTaxonomyTranslationPEO
- ApplTaxonomyHierarchyPEO

The following public view objects are located in the oracle.apps.fnd.applcore.model.publicView package and are exposed by Applications Core:

- ApplTaxonomyPVO: The view object on top of ApplTaxonomyPEO. It has these view criteria exposed:
  - ProductLineCriteria: Restricts the output to a given Application Line as specified by the bind variable bProductLine. By default, the ProductLineCriteria will be applied on the view objects exposed on the view object by Taxonomy (public and private) with the value of 1 for the Oracle Fusion application line. If you need to get data for another application line, you can set the appropriate value for the bind variable bProductLine.
  - IsSeedDataAllowedCriteria: where
    ApplTaxonomyEO.IS_SEED_DATA_ALLOWED=:bIsSeedDataAllowed, where :bIsSeedDataAllowed is a named bind variable with a default value of N.
  - ApplicationCriteria: Restricts the output to Applications Module Type.
  - FamilyCriteria: Restricts the output to Family Module Type.
  - ModuleIdCriteria: Restricts the output to a given module ID as specified by the bind variable bModuleId.
- **ModuleTypeCriteria**: Restricts the output to a given moduleType as specified by the bind variable `bModuleType`.

- **ApplTaxonomyTranslationPVO**: The view object on top of `ApplTaxonomyTranslationPEO`. It has these view criteria exposed:
  - **ProductLineCriteria**: where `ApplTaxonomyEO.PRODUCT_LINE=:bProductLine` where `:bProductLine` is a named bind variable with a default value of 1 - the Oracle Fusion Productline.
  - **IsSeedDataAllowedCriteria**: where `ApplTaxonomyEO.IS_SEED_DATA_ALLOWED=:bIsSeedDataAllowed`, where `:bIsSeedDataAllowed` is a named bind variable with a default value of `N`.

- **ApplTaxonomyHierarchyPVO**: The view object on top of `ApplTaxonomyHierarchyPEO`. It has these view criteria exposed:
  - **ProductLineCriteria**: where `ApplTaxonomyEO.PRODUCT_LINE=:bProductLine` where `:bProductLine` is a named bind variable with a default value of 1 - the Oracle Fusion Productline.
  - **IsSeedDataAllowedCriteria**: where `ApplTaxonomyEO.IS_SEED_DATA_ALLOWED=:bIsSeedDataAllowed`, where `:bIsSeedDataAllowed` is a named bind variable with a default value of `N`.

- **ApplTaxonomyHierarchyFullPVO**: Has a join between the hierarchy table's `source_module_id` and the taxonomy table's `module_id`, and between the hierarchy table's `target_module_id` and the taxonomy table's `module_id`. This view object is rarely needed. It has these view criteria exposed:
  - **ProductLineCriteria**: where `ApplTaxonomyEO.PRODUCT_LINE=:bProductLine` where `:bProductLine` is a named bind variable with a default value of 1 - the Oracle Fusion Productline.
  - **IsSeedDataAllowedCriteria**: where `ApplTaxonomyEO.IS_SEED_DATA_ALLOWED=:bIsSeedDataAllowed`, where `:bIsSeedDataAllowed` is a named bind variable with a default value of `N`.

- **ApplicationPVO**: This view object is shaped to match the Application view object that was on top of the FND_APPLICATIONS view. Note that `FND_APPLICATIONS`, which was a table in R12, has now been changed to a view on top of Taxonomy tables. It has these view criteria exposed:
  - **ProductLineCriteria**: where `ApplTaxonomyEO.PRODUCT_LINE=:bProductLine` where `:bProductLine` is a named bind variable with a default value of 1 - the Oracle Fusion Productline.
  - **IsSeedDataAllowedCriteria**: where `ApplTaxonomyEO.IS_SEED_DATA_ALLOWED=:bIsSeedDataAllowed`, where `:bIsSeedDataAllowed` is a named bind variable with a default value of `N`.
• **ApplTaxonomyFullDeliveryPVO** — This view object is a join between the Hierarchy table’s `source_module_id` and the Taxonomy table’s `module_id`. When a self-referential view link is created on this view object between the `source_module_id` and `target_module_id` (1..*), it can be used to traverse the taxonomy delivery hierarchy. For more details, see the non-public view object `ApplTaxonomyFullDeliveryVO` and its view link. It has these view criteria exposed:

  - **ProductLineCriteria**: where
    
    ```sql
    ApplTaxonomyEO.PRODUCT_LINE=:bProductLine
    where :bProductLine is a named bind variable with a default value of 1 – the Oracle Fusion Productline.
    ```

  - **IsSeedDataAllowedCriteria**: where
    
    ```sql
    ApplTaxonomyEO.IS_SEED_DATA_ALLOWED=:bIsSeedDataAllowed,
    where :bIsSeedDataAllowed is a named bind variable with a default value of N.
    ```

• **ApplTaxonomySeedDataPVO** — Used for extracting taxonomy seed data. It has these view criteria exposed:

  - **ProductLineCriteria**: where
    
    ```sql
    ApplTaxonomyEO.PRODUCT_LINE=:bProductLine
    where :bProductLine is a named bind variable with a default value of 1 – the Oracle Fusion Productline.
    ```

  - **IsSeedDataAllowedCriteria**: where
    
    ```sql
    ApplTaxonomyEO.IS_SEED_DATA_ALLOWED=:bIsSeedDataAllowed,
    where :bIsSeedDataAllowed is a named bind variable with a default value of N.
    ```

• **ApplTaxonomyApplicationPVO**

  - **ProductLineCriteria**: where
    
    ```sql
    ApplTaxonomyEO.PRODUCT_LINE=:bProductLine
    where :bProductLine is a named bind variable with a default value of 1 – the Oracle Fusion Productline.
    ```

  - **IsSeedDataAllowedCriteria**: where
    
    ```sql
    ApplTaxonomyEO.IS_SEED_DATA_ALLOWED=:bIsSeedDataAllowed,
    where :bIsSeedDataAllowed is a named bind variable with a default value of Y.
    ```

### 68.2.3.1 Access the Entity and View Objects

To access the Entity and View objects, and other taxonomy components, create a new **File System Connection** to `adflib`:

1. In the **Resource Palette**, open the folder icon and select **New Connection > File System** as shown in **Figure 68-2**:
2. Configure the new connection so it resembles the example in Figure 68-3:

**Figure 68-3 Configuring a New File System Connection**

3. The new connection will display in the Connections tree, shown in Figure 68-4:
4. The Entity and View objects are located in Taxonomy-Model.jar > Business Components, shown in Figure 68-5:

![Diagram](image1.png)

**Figure 68-5  Locating the Entity and View Objects in the Connections Tree**

### 68.2.4 How to Use Exposed Service Methods

**Note:**

- The service methods use the default Oracle Fusion Application Line value of 1. The application module APIs that do not accept an application line id assume that the data is being queried for the Oracle Fusion application line unless the API accepts the Taxonomy table primary key moduleId.

- The package for the private view objects and the application module conforms to Applications Packaging standards.

  The application module is
  `oracle.apps.fnd.applcore.taxonomy.taxonomyService.applicationModule`.

  The view objects module is
  `oracle.apps.fnd.applcore.taxonomy.taxonomyService.view`.

Two methods are exposed in the ApplTaxonomyAMImpl class:

- Given a module Id, this returns the taxonomy node:
  ```java
  public ApplTaxonomyFullDeliveryVORowImpl getTaxonomyModule(Raw moduleId);
  ```

- Given a moduleType, this returns an array of taxonomy nodes:
public ApplTaxonomyFullDeliveryVORowImpl[] getTaxonomyModules(String moduleType);

To access a taxonomy application module:

Note:
The ApplTaxonomyAMImpl package is oracle.apps.fnd.applcore.taxonomy.taxonomyService.applicationModule.

ApplTaxonomyAMImpl am = (ApplTaxonomyAMImpl)OAApplicationModuleImpl.getFNDNestedService(OAConstants.TAXONOMY_SERVICE, myAM.getDBTransaction());

Where myAM is the application module that you are working with. You can also create an instance of the ApplTaxonomyAMImpl class directly as needed.

To access a taxonomy node for a given application module, you can call the getTaxonomyModule() API on the module ID:

```
ApplTaxonomyFullDeliveryVORowImpl row = am.getTaxonomyModule(new oracle.jbo.domain.Raw("025000"));
```

To access the module name for that node, you can make a call to getModuleName():

```
String moduleName = row.getModuleName();
```

To access a set of taxonomy nodes for a given module type, you can call the getTaxonomyModules() API:

```
ApplTaxonomyFullDeliveryVORowImpl[] rows = am.getTaxonomyModules("FAMILY");
```

The following methods also are exposed:

- Given an application short name, return the application ID:
  getApplicationId(String shortName);

- Given an application ID, return the application short name:
  getApplicationShortName(int appid);

These APIs work if the existing data in FND_APPLICATIONS has been migrated to the Oracle Application Taxonomy tables.

68.2.5 How to Traverse the Taxonomy Hierarchy

The taxonomy delivery hierarchy can be navigated using the accessors for the children and parents.

To obtain the children of the current node, call getFirstChildApplTaxonomyFullDeliveryVO(), as shown in Example 68-1:

To obtain the parent of the current node, call getParentChildApplTaxonomyFullDeliveryVO(), as shown in Example 68-2.

Example 68-1 Obtaining the Children of the Current Node

```
RowIterator children = row getChildApplTaxonomyFullDeliveryVO();
while (children.hasNext())
{
    ApplTaxonomyFullDeliveryVORowImpl childRow =
```
68.3 Understanding Taxonomy MBeans

Taxonomy MBeans are useful for obtaining information about Oracle Fusion taxonomy, such as domain, application family, application, modules (UI, SOA, Webservices), and admin log configuration. These MBeans are available as Domain Runtime MBeans in WebLogic Server and expose several APIs, each of which provides specific information about the taxonomy of a deployed Oracle Fusion environment. These MBeans are consumed by application teams as utility APIs to verify information about their applications, and are also integrated with other applications. For instance Enterprise Manager for Oracle Fusion uses these APIs for building the discovery user interfaces. Taxonomy MBeans are registered into the application-defined MBeans after the administration server startup.

**Types of Taxonomy MBeans**

Two types of Taxonomy MBeans are available:

- **Topology MBean**: Provides information about the topology of the Oracle Fusion environment. Topology MBean data is dependent on the domain against which it is tested. In a particular domain, such as Setup, you can have more than one Product Family.

- **Log Configuration MBean**: Provides utilities for configuring logs at both User- and Site-level in an application.

MBeans as viewed from Enterprise Manager are shown in Figure 68-6.

**Figure 68-6  MBeans Viewed from Enterprise Manager**

![Topology MBean](image)

**Topology MBean Details:**

- Configuration MBeans
- Runtime MBeans
- Application-Defined MBeans
  - Environment
  - OracleApps
  - oracle.apps.fnd.appcore.diagnostics.config.mbean
  - oracle.apps.fnd.appcore.diagnostics.runtime.mbean
  - Oracle Node List
  - OracleAppsInfo
  - Domain: DefaultDomain
    - ConfigurationRuntimeMBean
    - TopologyRuntimeMBean

- More...
Attributes exposed by Topology MBeans are shown in Table 68-4.

**Table 68-4  Attributes Exposed by Topology MBeans**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllProductFamilyAndDomains</td>
<td>Gets all domains and product families of an Oracle Fusion instance.</td>
<td>R</td>
</tr>
<tr>
<td>ConfigMBean</td>
<td>If true, it indicates that this is a Config MBean.</td>
<td>R</td>
</tr>
<tr>
<td>CurrentDomain</td>
<td>Gets the current domain.</td>
<td>R</td>
</tr>
<tr>
<td>CurrentPillarInfo</td>
<td>Get the current pillar.</td>
<td>R</td>
</tr>
<tr>
<td>CurrentPillarInstanceInfo</td>
<td>Returns all information about current PillarInstanceName.</td>
<td>R</td>
</tr>
<tr>
<td>CurrentProductFamily</td>
<td>Gets the current product family module key list.</td>
<td>R</td>
</tr>
<tr>
<td>CurrentProductFamilyInfo</td>
<td>Gets the current product family list information.</td>
<td>R</td>
</tr>
<tr>
<td>eventProvider</td>
<td>If true, it indicates that this MBean is an event provider as defined by JSR-77.</td>
<td>R</td>
</tr>
<tr>
<td>eventTypes</td>
<td>All the event types emitted by this MBean.</td>
<td>R</td>
</tr>
<tr>
<td>ListOfProducts</td>
<td>Gets the list of products for the current product family.</td>
<td>R</td>
</tr>
<tr>
<td>objectName</td>
<td>The MBean's unique JMX name.</td>
<td>R</td>
</tr>
<tr>
<td>PillarDBInfo</td>
<td>Get the database information for a particular pillar.</td>
<td>R</td>
</tr>
<tr>
<td>Pillars</td>
<td>Get all the pillars.</td>
<td>R</td>
</tr>
<tr>
<td>ProductFromEachProductFamily</td>
<td>Gets list of products for each product family.</td>
<td>R</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>If true, this MBean is read-only.</td>
<td>R</td>
</tr>
<tr>
<td>RestartNeeded</td>
<td>Indicates whether a restart is needed.</td>
<td>R</td>
</tr>
<tr>
<td>stateManageable</td>
<td>If true, it indicates that this MBean provides state management capabilities as defined by JSR-77.</td>
<td>R</td>
</tr>
<tr>
<td>statisticsProvider</td>
<td>If true, it indicates that this MBean in a statistic provider as defined by JSR-77.</td>
<td>R</td>
</tr>
<tr>
<td>SystemMBean</td>
<td>If true, it indicates that this MBean is a System MBean.</td>
<td>R</td>
</tr>
</tbody>
</table>

The operations exposed by Topology MBeans are shown in Table 68-5.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameters</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAllDeployedAppsInfo</td>
<td>Complete information on the deployed application for a particular product.</td>
<td>1</td>
<td>Array of javax.management.openmbean.TabularData</td>
</tr>
<tr>
<td>getAllEnterpriseAppsInfo</td>
<td>Gets the list of application information for a given product.</td>
<td>2</td>
<td>Array of javax.management.openmbean.TabularData</td>
</tr>
<tr>
<td>getAppListFromDeployedDomain</td>
<td>Get the application information from the deployed domain name.</td>
<td>1</td>
<td>Array of javax.management.openmbean.TabularData</td>
</tr>
<tr>
<td>getDependentApps</td>
<td>Returns dependent applications information on an application from its AppShortName.</td>
<td>1</td>
<td>Array of javax.management.openmbean.TabularData</td>
</tr>
<tr>
<td>getDependentMWComponents</td>
<td>Gets Dependent MW Components for the AppShortName.</td>
<td>1</td>
<td>Array of javax.management.openmbean.TabularData</td>
</tr>
<tr>
<td>getDeployedAppInfo</td>
<td>Complete information on the deployed application given for an Application Short Name.</td>
<td>1</td>
<td>Array of javax.management.openmbean.TabularData</td>
</tr>
<tr>
<td>getDeployedDomainFromLogicalDomain</td>
<td>Gets DeployedDomain information from the LogicalDomain name.</td>
<td>1</td>
<td>Array of javax.management.openmbean.TabularData</td>
</tr>
<tr>
<td>getDeployedDomainFromPillar</td>
<td>Get the deployed domain information of a particular type for a pillar.</td>
<td>2</td>
<td>Array of javax.management.openmbean.TabularData</td>
</tr>
<tr>
<td>getDeployedDomainInfo</td>
<td>Returns a map of domain information for a given domain name.</td>
<td>1</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>getDeployedDomainByCompositeName</td>
<td>Get the deployed domain list for a given composite name.</td>
<td>1</td>
<td>Array of javax.management.openmbean.TabularData</td>
</tr>
<tr>
<td>getDeployedDomainsByEnvironment</td>
<td>Returns the list of domains for a particular environment. If the EnvironmentShortName is null, it will return all the deployed domains.</td>
<td>1</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>getDomainnames</td>
<td>Gets the list of domain names for a given application short name.</td>
<td>1</td>
<td>Array of java.lang.String</td>
</tr>
<tr>
<td>getEndPointInfo</td>
<td>Gets the external and internal end points for a given application short name.</td>
<td>1</td>
<td>Array of javax.management.openmbean.TabularData</td>
</tr>
</tbody>
</table>
Table 68-5  (Cont.) Operations Exposed by Topology MBeans

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameter(s)</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>getEndPointInfoFromModule</td>
<td>Gets the domain external and internal domain end points for a Logical Module Name.</td>
<td>1</td>
<td>Array of javax.management.openmbean.TabularData</td>
</tr>
<tr>
<td>getEnterpriseAppName</td>
<td>Gets the list of application information for a given application short name.</td>
<td>1</td>
<td>javax.management.openmbean.TabularData</td>
</tr>
<tr>
<td>getEssApplicationInfo</td>
<td>Gets ESS application information for a given product family module id.</td>
<td>1</td>
<td>javax.management.openmbean.TabularData</td>
</tr>
<tr>
<td>getLbasInfo</td>
<td>Gets the list of LBA information for the given product family.</td>
<td>1</td>
<td>Array of javax.management.openmbean.TabularData</td>
</tr>
<tr>
<td>getListOfDeployedApps</td>
<td>Gets the list of deployed applications for a particular product.</td>
<td>1</td>
<td>Array of java.lang.String</td>
</tr>
<tr>
<td>getListOfDomains</td>
<td>Gets the list of domain names for a given product family</td>
<td>1</td>
<td>Array of java.lang.String</td>
</tr>
<tr>
<td>getListOfEnterpriseApps</td>
<td>Gets the list of applications for a given product.</td>
<td>2</td>
<td>Array of java.lang.String</td>
</tr>
<tr>
<td>getListOfLbas</td>
<td>Gets the list of LBA (module names) for the given product family.</td>
<td>1</td>
<td>Array of java.lang.String</td>
</tr>
</tbody>
</table>

Log Configuration MBeans

Log Configuration MBean Details:

MBean Name oracle.topology:name=LogConfiguration,type=LogConfigurationRuntimeMBean
Description Log Configuration MBean APIs

Attributes exposed by Log Configuration MBeans are shown in Table 68-6.

Table 68-6  Attributes Exposed by Log Configuration MBeans

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfigMBean</td>
<td>If true, it indicates that this is a Config MBean.</td>
<td>R</td>
</tr>
<tr>
<td>eventProvider</td>
<td>If true, it indicates that this MBean is an event provider as defined by JSR-77.</td>
<td>R</td>
</tr>
<tr>
<td>eventTypes</td>
<td>All the event types emitted by this MBean.</td>
<td>R</td>
</tr>
<tr>
<td>LogConfigInformation</td>
<td>Gets the log configuration information at Site level.</td>
<td>R</td>
</tr>
<tr>
<td>objectname</td>
<td>The MBean’s unique JMX name.</td>
<td>R</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>If true, this MBean is read-only.</td>
<td>R</td>
</tr>
<tr>
<td>RestartNeeded</td>
<td>Indicates whether a restart is needed.</td>
<td>R</td>
</tr>
</tbody>
</table>
Table 68-6  (Cont.) Attributes Exposed by Log Configuration MBeans

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>stateManageable</td>
<td>If true, it indicates that this MBean provides State Management capabilities as defined by JSR-77.</td>
<td>R</td>
</tr>
<tr>
<td>statisticsProvider</td>
<td>If true, it indicates that this MBean in a statistic provider as defined by JSR-77.</td>
<td>R</td>
</tr>
<tr>
<td>SystemMBean</td>
<td>If true, it indicates that this MBean is a System MBean.</td>
<td>R</td>
</tr>
</tbody>
</table>

Operations exposed by Log Configuration MBeans are shown in Table 68-7.

Table 68-7  Operations Exposed by Log Configuration MBeans

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameter(s)</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>addUserLogConfig</td>
<td>Adds the log configuration information for a particular user.</td>
<td>8</td>
<td>boolean</td>
</tr>
<tr>
<td>deleteUserLogConfig</td>
<td>Delete the log configuration information for a particular user.</td>
<td>1</td>
<td>void</td>
</tr>
<tr>
<td>editUserLogConfig</td>
<td>Edit the log configuration information for a particular user.</td>
<td>8</td>
<td>boolean</td>
</tr>
<tr>
<td>getUserInfo</td>
<td>Gets the user information (the GUID) for users either having or not having the log configuration information.</td>
<td>2</td>
<td>javax.management.openmbean.TabularData</td>
</tr>
<tr>
<td>getUserLogConfigInformatio</td>
<td>Gets the log configuration information for a particular user.</td>
<td>1</td>
<td>Array of javax.management.openmbean.TabularData</td>
</tr>
<tr>
<td>updateLogConfigInformation</td>
<td>Update the log configuration information at Site level.</td>
<td>8</td>
<td>void</td>
</tr>
</tbody>
</table>

Sample Code to Invoke MBean APIs

Sample code is shown in Example 68-3.

Example 68-3  Sample Code to Invoke MBean APIs

```java
String appShortName  = "<ESS/Service Name>";
MBeanServerConnection serverCon =
    ServiceLocator.getInstance().getServerConnection(jmxURL, jmxKey);
ObjectName serviceMBean = new ObjectName("oracle.topology:Location=DefaultServer,name=Topology,
    type=TopologyRuntimeMBean,Application=Topology");
TabularData domainInfo = (TabularData) serverCon.invoke(serviceMBean,
    "getEndPointInfo",
    new Object[]{appShortName},
    new String[]{String.class.getName() } );
Collection domainProps = domainInfo.values();
Iterator iterator = domainProps.iterator();
while (iterator.hasNext())
```
{CompositeData data = (CompositeData) iterator.next();
String AppShortName = (String) data.get("AppShortName");
String CloudName = (String) data.get("CloudName");
String DeployedDomainName = (String) data.get("DeployedDomainName");
String LogicalDomainName= (String) data.get("LogicalDomainName");
String InternalEndPoint = (String) data.get("InternalEndPoint");
String ExternalEndPoint = (String) data.get("ExternalEndPoint");
String AdminEndPoint = (String) data.get("AdminEndPoint");
}
This appendix provides a reference for the commands available for the Oracle Enterprise Crawl and Search (ECSF) Command Line Administration Utility. You can use the ECSF Command Line Administration Utility to quickly test and manage the searchable objects without having to use Oracle Enterprise Manager Fusion Applications Control for ECSF.

### 69.1 ECSF Command Line Administration Utility Commands

Table 69-1 shows the commands you can use to administer search. The commands appear in alphabetical order.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activate object ID</td>
<td>Activates a searchable object so that a query of it returns results. Specify the ID number corresponding to the searchable object you want to activate.&lt;br&gt;Only a searchable object that has been deployed can be activated.&lt;br&gt;Customized searchable objects cannot be activated using the ECSF Command Line Administration Utility. You must activate customized searchable objects by using the Fusion Applications Control for ECSF.</td>
</tr>
<tr>
<td>add object ID to category ID</td>
<td>Associates the searchable object you specify with the search category you specify. Specify the ID number corresponding to the searchable object you want to add to the search category, and specify the ID number corresponding to the search category to which you want to add the searchable object.&lt;br&gt;Searchable objects must be deployed before you can associate them with search categories. Search categories must be undeployed before you can associate searchable objects with them. You can associate the same searchable object with multiple search categories. You must issue the command while managing the search engine instance with which the search category is associated.</td>
</tr>
</tbody>
</table>
### ECSF Command Line Administration Utility Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>add object to category ID</strong></td>
<td>Associates a searchable object with the search category you specify. Specify the ID number corresponding to the search category to which you want to add the searchable object. The ECSF Command Line Administration Utility displays a list of available searchable objects and prompts you to enter the ID corresponding to the searchable object you want to associate with the search category. Searchable objects must be deployed before you can associate them with search categories. Search categories must be undeployed before you can associate searchable objects with them. You can associate the same searchable object with multiple search categories. You must issue the command while managing the search engine instance with which the search category is associated.</td>
</tr>
<tr>
<td><strong>add object ID to schedule ID</strong></td>
<td>Associates the searchable object you specify with the index schedule you specify. Specify the ID number corresponding to the searchable object you want to add to the index schedule, and specify the ID number corresponding to the index schedule to which you want to add the searchable object. Searchable objects must be deployed before you can associate them with index schedules. You can only associate each searchable object with one index schedule. Only a searchable object that is not already associated with an index schedule can be added to an index schedule. Index schedules must be undeployed before you can associate searchable objects with them. You must issue the command while managing the search engine instance with which the index schedule is associated.</td>
</tr>
<tr>
<td><strong>add object to schedule ID</strong></td>
<td>Associates the searchable object you specify with the index schedule you specify. Specify the ID number corresponding to the index schedule to which you want to add the searchable object. The ECSF Command Line Administration Utility displays a list of available searchable objects and prompts you to enter the ID corresponding to the searchable object you want to associate with the index schedule. Searchable objects must be deployed before you can associate them with index schedules. You can only associate each searchable object with one index schedule. Only a searchable object that is not already associated with an index schedule can be added to an index schedule. Index schedules must be undeployed before you can associate searchable objects with them. You must issue the command while managing the search engine instance with which the index schedule is associated.</td>
</tr>
<tr>
<td><strong>add unassigned object to instance</strong></td>
<td>Associates a searchable object with the specified search engine instance. The ECSF Command Line Administration Utility displays a list of available searchable objects and prompts you to enter the ID corresponding to the searchable object you want to add to the search engine instance. You must issue the command while managing the search engine instance to which you want to add the searchable object. A searchable object can only be associated with one search engine instance at a time.</td>
</tr>
<tr>
<td><strong>add unassigned object ID to instance</strong></td>
<td>Associates a searchable object with the specified search engine instance. Specify the ID corresponding to the searchable object you want to add to the search engine instance. You must issue the command while managing the search engine instance to which you want to add the searchable object. A searchable object can only be associated with one search engine instance at a time.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>connect to database</td>
<td>Creates the connection to a database using a system identifier (SID). Follow the prompts to enter a username and password, as well as field values.</td>
</tr>
<tr>
<td>connect to database hostname port SID</td>
<td>Creates the connection to a database using a system identifier (SID). Specify the host name, port number, and SID. Follow the prompts to enter a username and password.</td>
</tr>
<tr>
<td>connect to database descriptor</td>
<td>Creates the connection to a database using a database descriptor. Follow the prompts to enter a username and password, as well as field values.</td>
</tr>
<tr>
<td>connect to database descriptor 'descriptor'</td>
<td>Creates the connection to a database using a database descriptor. Specify the descriptor, enclosing it in quotes, with either the system identifier (SID) or service name, for example: Using SID: ' (DESCRIPTION= (ADDRESS= (PROTOCOL= tcp) (HOST= fusionhost123) (PORT=5521)) (CONNECT_DATA= (SID= dbmsdb2)))' Using service name: ' (DESCRIPTION= (ADDRESS= (PROTOCOL= tcp) (HOST= fusionhost123) (PORT=5521)) (CONNECT_DATA= (SERVICE_NAME= myservice)))' Follow the prompts to enter a username and password.</td>
</tr>
<tr>
<td>connect to database service</td>
<td>Creates the connection to a database using a service name. Follow the prompts to enter a username and password, as well as field values.</td>
</tr>
<tr>
<td>connect to database service host name port servicename</td>
<td>Creates the connection to a database using a service name. Specify the host name, port number, and service name. Follow the prompts to enter a username and password.</td>
</tr>
<tr>
<td>connect to mbeanserver</td>
<td>Creates the connection to an MBean server. Follow the prompts to enter a username and password, as well as field values.</td>
</tr>
<tr>
<td>connect to mbeanserver hostname port</td>
<td>Creates the connection to an MBean server. Specify the host name and port number. Follow the prompts to enter a username and password.</td>
</tr>
<tr>
<td>create category</td>
<td>Adds a new search category to the ECSF_SEARCH_INDEX_GROUP table in the Oracle Fusion Applications database. Follow the prompts to enter field values. If you issue the command while managing a search engine instance, the search category is automatically associated with the search engine instance you are managing. If you issue the command outside a search engine instance context, the ECSF Command Line Administration Utility displays a list of the available search engine instances and prompts you to choose a search engine instance for the search category you want to create.</td>
</tr>
</tbody>
</table>
Table 69-1 (Cont.) ECSF Command Line Administration Utility Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>create category set</td>
<td>Adds a new search category to the ECSF_SEARCH_INDEX_GROUP table in the Oracle Fusion Applications database. Directly pass in field name-value pairs with the command. The field names and values must be enclosed in quotes. If the field name or value contains a quote, escape it with a backslash, for example, &quot;field name with &quot;escaped&quot; quotes&quot;. If you issue the command while managing a search engine instance, the search category is automatically associated with the search engine instance you are managing. If you issue the command outside of a search engine instance context, the ECSF Command Line Administration Utility displays a list of the available search engine instances and prompts you to choose a search engine instance for the search category you want to create.</td>
</tr>
<tr>
<td>create instance</td>
<td>Adds a new search engine instance to the specified search engine type. Follow the prompts to enter field values. You must issue the command while not managing a search engine instance.</td>
</tr>
<tr>
<td>create instance set</td>
<td>Adds a new search engine instance to the specified search engine type. Directly pass in field name-value pairs with the command. The field names and values must be enclosed in quotes. If the field name or value contains a quote, escape it with a backslash, for example, &quot;field name with &quot;escaped&quot; quotes&quot;. You must issue the command while not managing a search engine instance.</td>
</tr>
<tr>
<td>create object</td>
<td>Adds a new searchable object to the specified search engine type. Follow the prompts to enter field values. If you issue the command while managing a search engine instance, the searchable object is automatically associated with the search engine instance you are managing. If you issue the command outside a search engine instance context, the ECSF Command Line Administration Utility displays a list of the available search engine instances and prompts you to choose a search engine instance for the searchable object you want to create.</td>
</tr>
<tr>
<td>create schedule</td>
<td>Adds a new index schedule to the ECSF_INDEX_SCHEDULE table in the Oracle Fusion Applications database. Follow the prompts to enter field values. If you issue the command while managing a search engine instance, the index schedule is automatically associated with the search engine instance you are managing. If you issue the command outside a search engine instance context, the ECSF Command Line Administration Utility displays a list of the available search engine instances and prompts you to choose a search engine instance for the index schedule you want to create.</td>
</tr>
<tr>
<td>create schedule set</td>
<td>Adds a new index schedule to the ECSF_INDEX_SCHEDULE table in the Oracle Fusion Applications database. Directly pass in field name-value pairs with the command. The field names and values must be enclosed in quotes. If the field name or value contains a quote, escape it with a backslash, for example, &quot;field name with &quot;escaped&quot; quotes&quot;. If you issue the command while managing a search engine instance, the index schedule is automatically associated with the search engine instance you are managing. If you issue the command outside a search engine instance context, the ECSF Command Line Administration Utility displays a list of the available search engine instances and prompts you to choose a search engine instance for the index schedule you want to create.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>create unassigned object</td>
<td>Adds a new searchable object to the specified search engine type. Follow the prompts to enter field values. The searchable object is not associated with a search engine instance.</td>
</tr>
<tr>
<td>deactivate object ID</td>
<td>Deactivates a searchable object so that a query of it does not return results. Specify the ID number corresponding to the searchable object you want to deactivate. Only an activated searchable object can be deactivated. Deactivated searchable objects are still available for the search engine instance to crawl.</td>
</tr>
<tr>
<td>delete category ID</td>
<td>Disassociates the specified search category from the search engine instance and removes it from the ECSF_SEARCH_INDEX_GROUP table in the Oracle Fusion Applications database. Specify the ID number corresponding to the search category you want to delete. You must issue the command while managing the search engine instance with which the search category is associated.</td>
</tr>
<tr>
<td>delete external category ID</td>
<td>Removes the specified external search category from the ECSF_SEARCH_INDEX_GROUP table in the Oracle Fusion Applications database and makes it unavailable for querying. Specify the ID number of the external searchable category you want to delete.</td>
</tr>
<tr>
<td>delete instance ID</td>
<td>Removes the specified search engine instance. You cannot delete search engine instances while you are managing an engine instance. You cannot delete a search engine instance if there are any deployed objects, categories, or schedules associated with it.</td>
</tr>
<tr>
<td>delete object ID</td>
<td>Removes the specified assigned searchable object (associated with a search engine instance) from the ECSF schema in the Oracle Fusion Applications database. You must issue the command while managing the search engine instance with which the searchable object is associated. If the searchable object has been deployed, you must undeploy it before you can delete its record from the database.</td>
</tr>
<tr>
<td>delete schedule ID</td>
<td>Disassociates the specified index schedule from the search engine instance and removes it from the ECSF_INDEX_SCHEDULE table in the Oracle Fusion Applications database. Specify the ID corresponding to the index schedule you want to delete. You must issue the command while managing the search engine instance with which the index schedule is associated.</td>
</tr>
<tr>
<td>delete unassigned object ID</td>
<td>Removes the specified unassigned searchable object (not associated with a search engine instance) from the ECSF schema in the Oracle Fusion Applications database.</td>
</tr>
<tr>
<td>deploy category ID</td>
<td>Deploys the specified search category to the search engine instance. Specify the ID number corresponding to the search category you want to deploy. Searchable objects must be associated with the search category before you can deploy it. You must issue the command while managing the search engine instance with which the search category is associated.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>deploy object ID</td>
<td>Deploy the searchable object you specify to the search engine instance to make the objects available for the search engine instance to crawl. Specify the ID number corresponding to the searchable object you want to deploy. The searchable objects deployed to the search engine instance must have a unique and fully qualified name, for example, oracle.apps.crm.Opportunity or oracle.apps.hcm.Opportunity. Only a searchable object that is associated with a search engine instance can be deployed.</td>
</tr>
<tr>
<td>deploy params for objects</td>
<td>Collectively updates all deployed searchable objects with the latest search engine instance parameters.</td>
</tr>
<tr>
<td>deploy params for object ID</td>
<td>Updates the specified searchable object with the latest search engine instance parameters.</td>
</tr>
<tr>
<td>deploy schedule ID</td>
<td>Deploys the specified index schedule to the search engine instance. Specify the ID number corresponding to the index schedule you want to deploy. Searchable objects must be associated with the index schedule before you can deploy it. You must issue the command while managing the search engine instance with which the index schedule is associated.</td>
</tr>
<tr>
<td>disconnect</td>
<td>Disconnects you from the current database or MBean server connection.</td>
</tr>
<tr>
<td>exit</td>
<td>Closes the ECSF Command Line Administration Utility.</td>
</tr>
<tr>
<td>help</td>
<td>Lists all the available help commands.</td>
</tr>
<tr>
<td>help activate</td>
<td>Lists the valid syntax for the activate command.</td>
</tr>
<tr>
<td>help add</td>
<td>Lists the valid syntax for the add commands.</td>
</tr>
<tr>
<td>help category</td>
<td>Lists the commands that can be used for search categories.</td>
</tr>
<tr>
<td>help connect</td>
<td>Lists the valid syntax for the connect commands</td>
</tr>
<tr>
<td>help create category</td>
<td>Lists the fields available for the create category commands.</td>
</tr>
<tr>
<td>help create instance</td>
<td>Lists the fields available for the create instance commands.</td>
</tr>
<tr>
<td>help create object</td>
<td>Lists the fields available for the create object command.</td>
</tr>
<tr>
<td>help create schedule</td>
<td>Lists the fields available for the create schedule command.</td>
</tr>
<tr>
<td>help create unassigned object</td>
<td>Lists the fields available for the create unassigned object command.</td>
</tr>
<tr>
<td>help create schedule</td>
<td>Lists the fields available for the create schedule commands.</td>
</tr>
<tr>
<td>help deactivate</td>
<td>Lists the valid syntax for the deactivate object command.</td>
</tr>
<tr>
<td>help delete</td>
<td>Lists the valid syntax for the delete commands.</td>
</tr>
<tr>
<td>help deploy</td>
<td>Lists the valid syntax for the deploy commands.</td>
</tr>
<tr>
<td>help disconnect</td>
<td>Lists the valid syntax for the disconnect command.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>help instance</td>
<td>Lists the commands that can be used for search engine instances.</td>
</tr>
<tr>
<td>help list</td>
<td>Lists the valid syntax for the list commands.</td>
</tr>
<tr>
<td>help manage</td>
<td>Lists the valid syntax for the manage commands.</td>
</tr>
<tr>
<td>help object</td>
<td>Lists the commands that can be used for searchable objects.</td>
</tr>
<tr>
<td>help param</td>
<td>Lists the commands that can be used for search engine instance parameters.</td>
</tr>
<tr>
<td>help remove</td>
<td>Lists the valid syntax for the remove commands.</td>
</tr>
<tr>
<td>help schedule</td>
<td>Lists the commands that can be used for index schedules.</td>
</tr>
<tr>
<td>help set</td>
<td>Lists the valid syntax for the set commands.</td>
</tr>
<tr>
<td>help showdetails</td>
<td>Lists the valid syntax for the showdetails commands.</td>
</tr>
<tr>
<td>help start</td>
<td>Lists the valid syntax for the start command.</td>
</tr>
<tr>
<td>help stop</td>
<td>Lists the valid syntax for the stop command.</td>
</tr>
<tr>
<td>help unassigned object</td>
<td>Lists the commands that can be used for unassigned searchable objects.</td>
</tr>
<tr>
<td>help undeploy</td>
<td>Lists the valid syntax for the undeploy commands.</td>
</tr>
<tr>
<td>help unmanage</td>
<td>Lists the valid syntax for the unmanage commands.</td>
</tr>
<tr>
<td>help update category</td>
<td>Lists the fields available for the update category commands.</td>
</tr>
<tr>
<td>help update instance</td>
<td>Lists the fields available for the update instance commands.</td>
</tr>
<tr>
<td>help update schedule</td>
<td>Lists the fields available for the update schedule commands.</td>
</tr>
<tr>
<td>help register object</td>
<td>Lists the fields available for the register object commands.</td>
</tr>
<tr>
<td>import external categories</td>
<td>Lists one by one all the external categories of the search engine instance you are managing and prompts you to confirm whether you want to import each external category. Enter Y to import the external category, which adds it to the ECSF_SEARCH_INDEX_GROUP table in the Oracle Fusion Applications database. Enter N to cancel the importing of the external category. The default value is N. All external search categories that have been previously imported will be replaced by the latest import from Oracle Secure Enterprise Search (Oracle SES). If you had previously deleted any of the records corresponding to the external search categories, you must delete them again to make them unavailable for querying.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>import external categories for instance ID</td>
<td>Lists one by one all the external categories of the search engine instance you specify and prompts you to confirm whether you want to import each external category. Enter Y to import the external category, which adds it to the ECSF_SEARCH_INDEX_GROUP table in the Oracle Fusion Applications database. Enter N to cancel the importing of the external category. The default value is N. All external search categories that have been previously imported will be replaced by the latest import from Oracle Secure Enterprise Search (Oracle SES). If you had previously deleted any of the records corresponding to the external search categories, you must delete them again to make them unavailable for querying.</td>
</tr>
<tr>
<td>list categories</td>
<td>Lists the search categories and their corresponding ID numbers for the search engine instance you are managing.</td>
</tr>
<tr>
<td>list categories for instance ID</td>
<td>Lists the search categories and their corresponding ID numbers for the search engine instance you specify. Specify the ID number corresponding to the desired search engine instance.</td>
</tr>
<tr>
<td>list external search categories</td>
<td>Lists the external search categories and their corresponding ID numbers for the search engine instance you are managing.</td>
</tr>
<tr>
<td>list external search categories for instance ID</td>
<td>Lists the external search categories and their corresponding ID numbers for the search engine instance you specify. Specify the ID number corresponding to the desired search engine instance.</td>
</tr>
<tr>
<td>list instances</td>
<td>Lists the search engine instances and their corresponding ID numbers</td>
</tr>
<tr>
<td>list objects</td>
<td>Lists a summary of the searchable objects associated with the search engine instance you are managing.</td>
</tr>
<tr>
<td>list objects for category ID</td>
<td>Lists a summary of the searchable objects associated with the search category you specify. Specify the ID number corresponding to the desired search category.</td>
</tr>
<tr>
<td>list objects for instance ID</td>
<td>Lists a summary of the searchable objects associated with the search engine instance you specify. Specify the ID number corresponding to the desired search engine instance.</td>
</tr>
<tr>
<td>list objects for schedule ID</td>
<td>Lists a summary of the searchable objects associated with the index schedule you specify. Specify the ID number corresponding to the desired index schedule.</td>
</tr>
<tr>
<td>list params for instance</td>
<td>Lists the parameters that are available for the set param command for the engine instance you are managing.</td>
</tr>
<tr>
<td>list params for instance ID</td>
<td>Lists the parameters that are available for the set param command for the search engine instance you specify. Specify the ID number corresponding to the desired search engine instance.</td>
</tr>
<tr>
<td>list schedules</td>
<td>Lists the index schedules associated with the search engine instance you are managing.</td>
</tr>
<tr>
<td>list schedules for instance ID</td>
<td>Lists the index schedules associated with the search engine instance you specify. Specify the ID number corresponding to the desired search engine instance.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>list unassigned objects</td>
<td>Lists the unassigned searchable objects (not associated with a search engine instance) and their corresponding ID numbers.</td>
</tr>
<tr>
<td>manage instance</td>
<td>Sets the context to the specified search engine instance. The ECSF Command Line Administration Utility lists all the search engine instances and their corresponding ID numbers and prompt you for the ID number of the search engine instance you want to manage.</td>
</tr>
<tr>
<td>manage instance ID</td>
<td>Sets the context to the search engine instance you specify. Specify the ID number corresponding to the search engine instance you want to manage.</td>
</tr>
<tr>
<td>register idplugin</td>
<td>Registers an identity plug-in for the instance you are managing. The deployment of the Federated Trust Entity occurs when the identity plug-in is registered.</td>
</tr>
<tr>
<td>register idplugin for instance ID</td>
<td>Registers an identity plug-in for the search engine instance you specify. Specify the ID number corresponding to the desired search engine instance. The deployment of the Federated Trust Entity occurs when the identity plug-in is registered.</td>
</tr>
<tr>
<td>register object</td>
<td>Associates the specified searchable object with the search engine instance you are managing and creates a new record for the searchable object in the ECSF schema of the Oracle Fusion Applications database. Follow the prompts to enter field values. For BO Name, you must enter a fully qualified view object name defined in your application.</td>
</tr>
<tr>
<td>register object set &quot;fieldname&quot;=&quot;value&quot;,&quot;fieldname&quot;=&quot;value&quot;...</td>
<td>Associates the specified searchable object with the search engine instance you are managing and creates a new record for the searchable object in the Oracle Fusion Applications database. Directly pass in field name-value pairs with the command.</td>
</tr>
<tr>
<td>The field names and values must be enclosed in quotes. If the field name or value contains a quote, escape it with a backslash, for example, &quot;field name with escaped&quot; quotes.</td>
<td></td>
</tr>
<tr>
<td>register unassigned object</td>
<td>Creates a new record of an unassigned searchable object (not associated with a search engine instance) in the Oracle Fusion Applications database. Follow the prompts to enter field values. For BO Name, you must enter a fully qualified view object name defined in your application.</td>
</tr>
<tr>
<td>register unassigned object set &quot;fieldname&quot;=&quot;value&quot;,&quot;fieldname&quot;=&quot;value&quot;...</td>
<td>Creates a new record of an unassigned searchable object (not associated with a search engine instance) in the Oracle Fusion Applications database. Directly pass in field name-value pairs with the command.</td>
</tr>
<tr>
<td>The field names and values must be enclosed in quotes. If the field name or value contains a quote, escape it with a backslash, for example, &quot;field name with escaped&quot; quotes.</td>
<td></td>
</tr>
<tr>
<td>remove object from category ID</td>
<td>Disassociates a searchable object from the search category you specify. Specify the ID number corresponding to the search category from which you want to disassociate the searchable object. The ECSF Command Line Administration Utility displays a list of searchable objects and prompts you to enter the ID corresponding to the searchable object you want to remove from the search category.</td>
</tr>
<tr>
<td>You must issue the command while managing the search engine instance with which the search category is associated. The searchable object is still available for association to other search categories.</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>remove object ID from category ID</td>
<td>Disassociates the specified searchable object from the search category you specify. Specify the ID number corresponding to the searchable object you want to remove from the search category. Specify the ID number corresponding to the search category from which you want to disassociate the searchable object. You must issue the command while managing the search engine instance with which the search category is associated. The searchable object is still available for association to other search categories.</td>
</tr>
<tr>
<td>remove object from instance</td>
<td>Disassociates a searchable object from the specified search engine instance and makes it available for association to another search engine instance. The ECSF Command Line Administration Utility displays a list of searchable objects and prompts you to enter the ID corresponding to the searchable object you want to remove from the search engine instance. To disassociate a searchable object from a search engine instance, both the object and the instance must be undeployed.</td>
</tr>
<tr>
<td>remove object from instance ID</td>
<td>Disassociates a searchable object from the specified search engine instance and makes it available for association to another search engine instance. Specify the ID number corresponding to the search engine instance from which you want to remove the searchable object. The ECSF Command Line Administration Utility displays a list of searchable objects and prompts you to enter the ID corresponding to the searchable object you want to remove from the search engine instance. To disassociate a searchable object from a search engine instance, both the object and the instance must be undeployed.</td>
</tr>
<tr>
<td>remove object ID from instance</td>
<td>Disassociates the specified searchable object from the search engine instance and makes it available for association to another search engine instance. Specify the ID number corresponding to the searchable object you want to remove. To disassociate a searchable object from a search engine instance, both the object and the instance must be undeployed.</td>
</tr>
<tr>
<td>remove object ID from instance ID</td>
<td>Disassociates the searchable object from the search engine instance and makes it available for association to another search engine instance. Specify the ID number corresponding to the searchable object you want to remove and the ID number corresponding to the search engine instance from which you want to remove the searchable object. To disassociate a searchable object from a search engine instance, both the object and the instance must be undeployed.</td>
</tr>
<tr>
<td>remove object from schedule ID</td>
<td>Disassociates a searchable object from the specified index schedule and makes it available to be added to another index schedule. Specify the ID number corresponding to the index schedule from which you want to disassociate the searchable object. The ECSF Command Line Administration Utility displays a list of searchable objects and prompts you to enter the ID corresponding to the searchable object you want to remove from the index schedule. To disassociate a searchable object from an index schedule, the index schedule must not be deployed. You must issue the command while managing the search engine instance with which the index schedule is associated.</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>remove object ID from schedule ID</td>
<td>Disassociates the searchable object you specify from the specified index schedule and makes it available to be added to another index schedule. Specify the ID number corresponding to the searchable object you want to disassociate and the ID number corresponding to the index schedule from which you want to disassociate the searchable object. To disassociate a searchable object from an index schedule, the index schedule must not be deployed. You must issue the command while managing the search engine instance with which the index schedule is associated.</td>
</tr>
<tr>
<td>set param &quot;paramname&quot;=&quot;value&quot;</td>
<td>Sets parameter values for the search engine instance. Use the following command syntax to directly pass in one parameter name-value pair at a time: See the &quot;Managing Search with Oracle Enterprise Crawl and Search Framework&quot; chapter in the Administrator's Guide for a list of known engine instance parameters. The parameter name and value must be enclosed in quotes. If the parameter name or value contains a quote, escape it with a backslash, for example, &quot;value with &quot;escaped&quot; quotes&quot;. You must issue the command while managing the search engine instance whose password parameters you want to set.</td>
</tr>
<tr>
<td>set password param</td>
<td>Sets password parameter values for the search engine instance. Pass in one password parameter and its password. See the &quot;Managing Search with Oracle Enterprise Crawl and Search Framework&quot; chapter in the Administrator's Guide for a list of known engine instance parameters. The password parameter name must be enclosed in quotes. If the parameter parameter name contains a quote, escape it with a backslash, for example, &quot;value with &quot;escaped&quot; quotes&quot;. You must issue the command while managing the search engine instance whose password parameters you want to set.</td>
</tr>
<tr>
<td>showdetails for category ID</td>
<td>Lists detailed information about the specified search category and the searchable objects associated with it.</td>
</tr>
<tr>
<td>showdetails for unassigned object ID</td>
<td>Lists the detailed information about the specified unassigned searchable object.</td>
</tr>
<tr>
<td>showdetails</td>
<td>Lists the detailed information for the search engine instance being managed. You must issue the command while managing a search engine instance.</td>
</tr>
<tr>
<td>showdetails for instance ID</td>
<td>Lists the detailed information about the specified search engine instance.</td>
</tr>
<tr>
<td>showdetails for object ID</td>
<td>Lists the detailed information about the specified searchable object.</td>
</tr>
<tr>
<td>showdetails for param ID</td>
<td>Lists the detailed information about the specified search engine instance parameter.</td>
</tr>
<tr>
<td>showdetails for schedule ID</td>
<td>Lists detailed information about the specified index schedule and the searchable objects associated with it.</td>
</tr>
</tbody>
</table>
### Table 69-1 (Cont.) ECSF Command Line Administration Utility Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start schedule ID</td>
<td>Launches the index schedule you specify and causes Oracle SES to create the full-text search indexes. Specify the ID corresponding to the index schedule you want to start.</td>
</tr>
<tr>
<td></td>
<td>Index schedules must be deployed to the search engine instance before you can start it. You must issue the command while managing the search engine instance with which the index schedule is associated.</td>
</tr>
<tr>
<td>stop schedule ID</td>
<td>Stops the specified index schedule that has been started and aborts the index process. Specify the ID number of the index schedule you want to stop. You must issue the command while managing the search engine instance with which the index schedule is associated.</td>
</tr>
<tr>
<td>undeploy category ID</td>
<td>Removes a search category from the search engine instance. Specify the ID number corresponding to the search category you want to undeploy. You must issue the command while managing the search engine instance with which the search category is associated.</td>
</tr>
<tr>
<td>undeploy object ID</td>
<td>Removes a searchable object from the search engine instance to make the object unavailable for the search engine instance to crawl. Specify the ID number corresponding to the searchable object you want to undeploy. Only deployed and deactivated searchable objects can be undeployed.</td>
</tr>
<tr>
<td>undeploy schedule ID</td>
<td>Removes the specified index schedule from the search engine instance. Specify the ID number corresponding to the index schedule you want to undeploy. You must issue the command while managing the search engine instance with which the index schedule is associated.</td>
</tr>
<tr>
<td>unmanage</td>
<td>Resets or exits the search engine instance context.</td>
</tr>
<tr>
<td>unmanage instance</td>
<td>Resets or exits the search engine instance context.</td>
</tr>
<tr>
<td>update category ID</td>
<td>Modifies the properties of the specified search category. Specify the ID number corresponding to the search category you want to modify. Follow the prompts to enter field values. Set the scope of the search category to GLOBAL to allow the search categories to be queried. You must issue the command while managing the search engine instance with which the search category is associated.</td>
</tr>
<tr>
<td>update category ID set</td>
<td>Modifies the properties of the specified search category. Use the following command syntax to directly pass in field name-value pairs with the command: The field names and values must be enclosed in quotes. If the field name or value contains a quote, escape it with a backslash, for example, &quot;field name with &quot;escaped&quot; quotes&quot;. Set the scope of the search category to GLOBAL to allow the search categories to be queried. You must issue the command while managing the search engine instance with which the search category is associated.</td>
</tr>
</tbody>
</table>

---

69-12 Oracle Fusion Applications Developer’s Guide
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>update external category ID</td>
<td>Modifies the application identity of the specified external search category. Specify the ID number corresponding to the external search category you want to modify. Follow the prompts to enter field values. Set the scope of the search category to GLOBAL to allow the search categories to be queried. You must issue the command while managing the search engine instance with which the external search category is associated.</td>
</tr>
<tr>
<td>update external category ID set &quot;fieldname&quot;=&quot;value&quot;</td>
<td>Modifies the application identity of the specified external search category. Use the following command syntax to directly pass in a field name-value pair with the command: The field name and value must be enclosed in quotes. If the field name or value contains a quote, escape it with a backslash, for example, &quot;field name with &quot;escaped&quot; quotes&quot;. Set the scope of the search category to GLOBAL to allow the search categories to be queried. You must issue the command while managing the search engine instance with which the external search category is associated.</td>
</tr>
<tr>
<td>update instance</td>
<td>Modifies the properties of the search engine instance you are currently managing. If you are not currently managing a search engine instance, the ECSF Command Line Administration Utility lists all the search engine instances and their corresponding ID numbers and prompt you for the ID number of the search engine instance you want to modify. Follow the prompts to enter field values.</td>
</tr>
<tr>
<td>update instance ID</td>
<td>Modifies the properties of the search engine instance you specify. Specify the ID number corresponding to the search engine instance you want to modify. Follow the prompts to enter field values.</td>
</tr>
<tr>
<td>update instance set &quot;fieldname&quot;=&quot;value&quot;,&quot;fieldname&quot;=&quot;value...&quot;</td>
<td>Modifies the properties of the search engine instance you are currently managing. Directly pass in field name-value pairs with the command. The field names and values must be enclosed in quotes. If the field name or value contains a quote, escape it with a backslash, for example, &quot;field name with &quot;escaped&quot; quotes&quot;.</td>
</tr>
<tr>
<td>update instance ID set &quot;fieldname&quot;=&quot;value&quot;,&quot;fieldname&quot;=&quot;value...&quot;</td>
<td>Modifies the properties of the search engine instance you specify. Specify the ID number corresponding to the search engine instance you want to modify and directly pass in field name-value pairs with the command. The field names and values must be enclosed in quotes. If the field name or value contains a quote, escape it with a backslash, for example, &quot;field name with &quot;escaped&quot; quotes&quot;.</td>
</tr>
<tr>
<td>update object ID</td>
<td>Modifies the display name and application ID of a deployed searchable object without first having to deactivate and undeploy the searchable object. Specify the ID number of the searchable object you want to modify. Follow the prompts to enter field values for the display name and application ID. You must issue the command while managing the search engine instance with which the searchable object is associated.</td>
</tr>
</tbody>
</table>
Table 69-1  (Cont.) ECSF Command Line Administration Utility Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>update object ID set &quot;fieldname&quot;=&quot;value&quot;,&quot;fieldname&quot;=&quot;value&quot;...</td>
<td>Modifies the display name and application ID of a deployed searchable object without first having to deactivate and undeploy the searchable object. Specify the ID number corresponding to the searchable object you want to modify and directly pass in field name-value pairs with the command. The field names and values must be enclosed in quotes. If the field name or value contains a quote, escape it with a backslash, for example, &quot;fieldname with &quot;escaped&quot; quotes&quot;. You must issue the command while managing the search engine instance with which the searchable object is associated.</td>
</tr>
<tr>
<td>update schedule ID</td>
<td>Modifies the properties of the index schedule you specify. Specify the ID number corresponding to the index schedule you want to modify. Follow the prompts to enter new field values.</td>
</tr>
<tr>
<td>update schedule ID set &quot;fieldname&quot;=&quot;value&quot;,&quot;fieldname&quot;=&quot;value&quot;...</td>
<td>Modifies the properties of the index schedule you specify. Specify the ID number corresponding to the index schedule you want to modify and directly pass in field name-value pairs with the command. The field names and values must be enclosed in quotes. If the field name or value contains a quote, escape it with a backslash, for example, &quot;fieldname with &quot;escaped&quot; quotes&quot;. You must issue the command while managing the search engine instance with which the index schedule is associated.</td>
</tr>
</tbody>
</table>
Configure Attachments Component Properties

This appendix provides information about the properties that you can use to change the default behavior of the Attachments, Attachments Carousel, and Attachment Views components user interfaces.

This appendix includes the following sections:

• About Configuring the Attachments Component UI
• About Configuring the Attachments Carousel UI
• About Configuring the Attachment Views Components UI
• About Configuring the Attachment Link Component UI

70.1 About Configuring the Attachments Component UI

The default behavior of the Attachments component can be changed by setting the properties on the component using the Property Inspector.

Following is a list of some of the properties that are supported by the Attachment component. The following table contains basic descriptions of why and how each property is used.

The most important properties to take note of are:

• mode - This property drives the UI that is rendered. It is automatically set when you add attachments to your page and should not be changed. Mode values include link, table, and columnLink.

• repositoryMode - This property indicates whether a user is aware of the content repository. Additional options appear for these users (repositoryMode=true), allowing them to perform advanced functions such as selecting repository files and folders to attach to the business object, and checking in and checking out attached files.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>String. Acceptable values are link, columnLink, and table.</td>
<td>The mode property is set automatically at the time the component is created.</td>
</tr>
</tbody>
</table>

Table 70-1 Attachments Component Properties
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RepositoryMode</td>
<td>Boolean: true / false. True: Repository related functionality is enabled,</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>including: the ability to select and attach a document/folder from the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Content Server, and version control.</td>
<td></td>
</tr>
<tr>
<td>AttachFolderAllowed</td>
<td>Not used.</td>
<td>N/A</td>
</tr>
<tr>
<td>AttachLatestVersion</td>
<td>Not used.</td>
<td>N/A</td>
</tr>
<tr>
<td>ApprovalEnabled</td>
<td>Boolean: true / false. True: Adds the Status column to the Attachments table.</td>
<td>False</td>
</tr>
<tr>
<td>DeleteMessage</td>
<td>String. Message to be displayed on Delete Attachment confirmation page.</td>
<td>Are you sure you want</td>
</tr>
<tr>
<td></td>
<td></td>
<td>to delete the selected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>attachment(s)?</td>
</tr>
<tr>
<td>Rows</td>
<td>Number. Maximum number of rows to display in a single range of rows in</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>attachment table. Some ranges may have fewer than the number specified here,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>i.e. the last range when there is an insufficient number of rows. To</td>
<td></td>
</tr>
<tr>
<td></td>
<td>display all rows at once, set this attribute value to 0.</td>
<td></td>
</tr>
<tr>
<td>FileTypeEnabled</td>
<td>Boolean: true / false. True: Display attachment type of &quot;File&quot; in the Type</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>list. False: the component is hidden.</td>
<td></td>
</tr>
<tr>
<td>TextTypeEnabled</td>
<td>Boolean: true / false. True: Display attachment type of &quot;Text&quot; in the Type</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>list.</td>
<td></td>
</tr>
<tr>
<td>UrlTypeEnabled</td>
<td>Boolean: true / false. True: Display attachment type of &quot;URL&quot; in the Type</td>
<td>True</td>
</tr>
<tr>
<td>NumAttachmentsDisplayed</td>
<td>Number. Maximum number of attachments displayed as links in the hover popup</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>when mode is set to link or columnLink.</td>
<td></td>
</tr>
<tr>
<td>AddAllowed</td>
<td>Boolean: true / false. Enables/disables the &quot;Add&quot; icon in the attachments</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>table toolbar and inline for link and columnLink modes. True: enables the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>icon in the toolbar. False: disables the icon in the toolbar.</td>
<td></td>
</tr>
<tr>
<td>AddEnabled</td>
<td>Boolean: true / false. True: shows the icon in the toolbar. False: hides</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>the icon in the toolbar.</td>
<td></td>
</tr>
<tr>
<td>ActionEntity</td>
<td>String. This must be an Attachment Entity Name. For use when showing the</td>
<td></td>
</tr>
<tr>
<td></td>
<td>attachments for multiple entities in the Attachments table/popup. Setting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>this property indicates that update and delete actions will only be</td>
<td></td>
</tr>
<tr>
<td></td>
<td>possible on this entity.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>UpdateAllowed</td>
<td>Boolean: <code>true</code> / <code>false</code>. Enables/disables the update icons in the attachments table toolbar. These include the &quot;Check In&quot;, &quot;Check Out&quot; and &quot;Cancel Check Out&quot; icons. This property also enables/disables the users' ability to update the Category, Title and Description of the Attachments. True: enables the icons and makes the Category, Title and Description columns editable for existing attachments. False: disables the update icons and makes the Category, Title and Description columns read-only for existing attachments.</td>
<td>True</td>
</tr>
<tr>
<td>UpdateEnabled</td>
<td>Boolean: <code>true</code> / <code>false</code>. Controls show/hide of the update icons in the attachments table toolbar. These include the &quot;Check In&quot;, &quot;Check Out&quot; and &quot;Cancel Check Out&quot; icons. True: shows the update icons in the toolbar. False: hides the update icons in the toolbar.</td>
<td>True</td>
</tr>
<tr>
<td>InsertMultiple</td>
<td>Boolean: <code>true</code> / <code>false</code>. Determines whether more than one attachment can be assigned. False: User can only add a one attachment.</td>
<td>True</td>
</tr>
<tr>
<td>DeleteAllowed</td>
<td>Boolean: <code>true</code> / <code>false</code>. Enables/disables the &quot;Delete&quot; icon in the attachments toolbar and inline for link and columnLink modes. True: Deleting attachments is allowed. True: enables the icon. False: disables the icon.</td>
<td>True</td>
</tr>
<tr>
<td>DeleteEnabled</td>
<td>Boolean: <code>true</code> / <code>false</code>. Controls show/hide of the &quot;Delete&quot; icon in the attachments table toolbar. True: shows the icon in the toolbar. False: hides the icon in the toolbar preventing users from deleting attachments.</td>
<td>True</td>
</tr>
<tr>
<td>ViewAllowed</td>
<td>Boolean: <code>true</code> / <code>false</code>. True: Viewing attachments is allowed. False: Links for viewing documents in Attachments tables are disabled.</td>
<td>True</td>
</tr>
<tr>
<td>DefaultCategory</td>
<td>String. Value selected as the default in the Category poplist when adding new attachments.</td>
<td></td>
</tr>
<tr>
<td>Rendered</td>
<td>Boolean: <code>true</code> / <code>false</code>. True: Component is rendered in the page. False: the component is not rendered.</td>
<td>True</td>
</tr>
<tr>
<td>Id</td>
<td>String. The identifier of the component.</td>
<td>Auto-generated</td>
</tr>
</tbody>
</table>
### Table 70-1 (Cont.) Attachments Component Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>Boolean: <code>true</code> / <code>false</code>. True: the component is displayed in the page. False: the component is hidden.</td>
<td>True</td>
</tr>
<tr>
<td>Description</td>
<td>String. Displays string as a tip when the user hovers the cursor over the Attachment component.</td>
<td></td>
</tr>
<tr>
<td>Label</td>
<td>String. Specifies the title for the Attachment popup.</td>
<td>Attachment</td>
</tr>
<tr>
<td>AutoHeightRows</td>
<td>Number. Sets the maximum number of rows that the table height will automatically adjust to depending on the amount of data in the table.</td>
<td>10</td>
</tr>
<tr>
<td>ContentDelivery</td>
<td>String. This property is only used with the AutoHeightRows property. If AutoHeightRows is not set, this property should be not be set either.</td>
<td>Immediate</td>
</tr>
<tr>
<td>RowBandingInterval</td>
<td>Number. The interval between which the row banding occurs. This value controls the display of the row banding in the table. For example, <code>rowBandingInterval=1</code> would display alternately banded rows in the grid.</td>
<td></td>
</tr>
<tr>
<td>ColumnBandingInterval</td>
<td>Number. The interval between which the column banding occurs. This value controls the display of the column banding in the table. For example, <code>columnBandingInterval=1</code> would display alternately banded columns in the grid.</td>
<td></td>
</tr>
<tr>
<td>ShowCategory</td>
<td>Boolean: <code>true</code> / <code>false</code>. Controls show/hide of the Category column in the Attachments table.</td>
<td>True</td>
</tr>
<tr>
<td>UpdateCategoryList</td>
<td>String. Stores a comma-separated list of Category Names that will be used at run time to populate Category LOV when a user adds new attachments or attempts to update the Category of an existing attachment. This list must be a subset of the Categories that are assigned to your Entity.</td>
<td></td>
</tr>
<tr>
<td>SecondaryToolbarRendered</td>
<td>Boolean: <code>true</code> / <code>false</code>. True: shows the secondary toolbar. False: hides the icon in the toolbar.</td>
<td>False</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>Boolean: <code>true</code> / <code>false</code>. True disables all creation/update/delete functions in the Attachment component at runtime.</td>
<td>False</td>
</tr>
<tr>
<td>attachmentsTrigger</td>
<td>String. For use when an Attachment component and Attachment Carousel component are on the same page. The Carousel component can be refreshed only when a change is made to the visible details in the Attachment component. Rather than being refreshed on every user event. Enter a space separated list of Attachment Carousel component IDs.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 70-1  (Cont.) Attachments Component Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>enableOrdering</td>
<td>String. Provides the end user with the option to control the order the attachment rows are displayed in the table. Set to <code>true</code> to enable.</td>
<td></td>
</tr>
<tr>
<td>enablePrimaryCategory</td>
<td>String. Users can designate one attachment to be the primary from all of the attachments assigned to a category. Set to <code>true</code> to enable.</td>
<td></td>
</tr>
<tr>
<td>UseResetActionListener</td>
<td>Boolean: <code>true</code> / <code>false</code>. True Activates the declarative af:resetActionListener which is associated with the OK and Cancel buttons for each popup in the Attachments component. The resetActionListener resets all of the values submitted to avoid the data in the form from being stale. For more information, see &quot;How to Use a Command Component to Reset Input Fields&quot; in Developing Web User Interfaces with Oracle ADF Faces.</td>
<td><code>false</code></td>
</tr>
<tr>
<td>fieldRenderOff</td>
<td>String. A space-separated list of fields that will not be rendered into the page. The fields are Attached By (attachedBy), Attached Date (attachedDate), Description, Shared, and Checked Out By (checkedOutBy).</td>
<td></td>
</tr>
<tr>
<td>fuse</td>
<td>Boolean: <code>true</code> / <code>false</code>. True renders the attachment component for a FUSE UI page. Only link mode is supported for FUSE.</td>
<td><code>false</code></td>
</tr>
<tr>
<td>confirmDelete</td>
<td>Boolean: <code>true</code> / <code>false</code>. True displays the Delete Attachment Confirmation page when the delete icon is used.</td>
<td><code>true</code></td>
</tr>
</tbody>
</table>

**Note:**

Activating the `enableOrdering` or `enablePrimaryCategory` properties on an existing page will require an update to the page bindings. Do the following:

1. Select the **Page Bindings** tab. In the Bindings list, select the Attachment Model.
2. Click on the link at the top of the page for the Page Definition File. The file will open with the Attachments collection selected.
3. At the bottom of the selection, add two new attributes so that the end of the entry looks like this:

```xml
<Item Value="SeqNum"/>
<Item Value="PrimaryCategoryFlag"/>
</AttrNames>
</nodeDefinition>
```
70.2 About Configuring the Attachments Carousel UI

The default behavior of the Attachments Carousel component can be changed by setting the properties on the component using the Property Inspector.

Note:
Most of the attributes map directly to either the Attachment, Carousel or CarouselItem attributes. Wherever there is overlap of attribute names, either Carousel or CarouselItem has been prefixed to assist in identifying which attribute is being set.

Following is a list of some of the properties that are supported by the Attachments Carousel component. Refer to Table 70-2 for basic descriptions of why and how each property is used.

The most important property to take note of is:

repositoryMode - This property indicates whether a user is aware of the content repository. Additional options appear for these users (repositoryMode=true), allowing them to perform advanced functions such as selecting repository files and folders to attach to the business object, and checking in and checking out attached files.

Table 70-2  Attachments Carousel Component Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String. The identifier of the component.</td>
<td>Auto-generated</td>
</tr>
<tr>
<td>Rendered</td>
<td>Boolean: true / false. True: Component is rendered in the page. False: Component is not rendered.</td>
<td>False</td>
</tr>
<tr>
<td>visible</td>
<td>Boolean: true / false. True: Component is displayed on the page. False: Component does not display on the page.</td>
<td>True</td>
</tr>
<tr>
<td>repositoryMode</td>
<td>Boolean: true / false. True: Repository-related functionality is enabled, including: the ability to select and attach a document/ folder from the Content Server, and version control.</td>
<td>False</td>
</tr>
<tr>
<td>fileTypeEnabled</td>
<td>Boolean: true / false. True: Display attachment type of “File” in the Type choice list. False: Component is hidden.</td>
<td>True</td>
</tr>
<tr>
<td>textTypeEnabled</td>
<td>Boolean: true / false. True: Display attachment type of “Text” in the Type choice list.</td>
<td>True</td>
</tr>
</tbody>
</table>
### Table 70-2  (Cont.) Attachments Carousel Component Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>urlTypeEnabled</td>
<td>Boolean: true / false. True: Display attachment type of &quot;URL&quot; in the Type choice list.</td>
<td>True</td>
</tr>
<tr>
<td>halign</td>
<td>String. Valid Values: start, center, end; determines the horizontal alignment of the carousel items.</td>
<td>center</td>
</tr>
<tr>
<td>orientation</td>
<td>String. Valid Values: horizontal, vertical horizontal (the default); for the images being displayed along an x-axis or vertical for y-axis.</td>
<td>horizontal</td>
</tr>
<tr>
<td>valign</td>
<td>String. Valid Values: middle, top, bottom; determines the vertical alignment of the carousel items.</td>
<td>middle</td>
</tr>
<tr>
<td>attachmentDetailsRendered</td>
<td>Boolean: true / false. True: shows the detail section, which provides information about the carousel item that has the current focus. False: hides the detail section.</td>
<td>False</td>
</tr>
<tr>
<td>carouselSecondaryToolbarRendered</td>
<td>Boolean: true / false. Enables/disables the &quot;Add&quot; icon in the attachments table toolbar and inline for link and columnLink modes. True: shows the carousel secondary toolbar. False: hides the icon in the toolbar.</td>
<td>True</td>
</tr>
<tr>
<td>carouselToolbarRendered</td>
<td>Boolean: true / false. True: shows the carousel toolbar. False: hides the icon in the toolbar.</td>
<td>True</td>
</tr>
<tr>
<td>secondaryToolbarRendered</td>
<td>Boolean: true / false. True: shows the secondary toolbar. False: hides the icon in the toolbar.</td>
<td>False</td>
</tr>
<tr>
<td>actionEntity</td>
<td>String. This must be an Attachment Entity Name. For use when showing the attachments for multiple entities in the Attachments table/popup. Setting this property indicates that update and delete actions will only be possible on this entity.</td>
<td></td>
</tr>
<tr>
<td>approvalEnabled</td>
<td>Boolean: true / false. True: Adds the Status column to the Attachments table.</td>
<td>False</td>
</tr>
<tr>
<td>approvalFunction</td>
<td>String. Not used</td>
<td></td>
</tr>
<tr>
<td>attachFolderAllowed</td>
<td>Boolean. Not used</td>
<td>N/A</td>
</tr>
<tr>
<td>attachLatestVersion</td>
<td>Boolean. Not used</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Table 70-2 (Cont.) Attachments Carousel Component Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>updateCategoryList</td>
<td>String. Stores a comma-separated list of Category Names that will be used at run time to populate Category LOV when a user adds new attachments or attempts to update the Category of an existing attachment. This list must be a subset of the Categories that are assigned to your Entity.</td>
<td></td>
</tr>
<tr>
<td>attachmentsTrigger</td>
<td>String. Enables the Attachment component to refresh other components in the page when user interaction updates the Attachment information. Works off a space separated list of component IDs.</td>
<td></td>
</tr>
<tr>
<td>autoHeightRows</td>
<td>Number. Sets the maximum number of rows that the table height will automatically adjust to depending on the amount of data in the table.</td>
<td>10</td>
</tr>
<tr>
<td>columnBandingInterval</td>
<td>Number. The interval between which the column banding occurs. This value controls the display of the column banding in the table. For example, columnBandingInterval=1 would display alternately banded columns in the grid.</td>
<td></td>
</tr>
<tr>
<td>contentDelivery</td>
<td>String. This property is only used with the AutoHeightRows property. If AutoHeightRows is not set, this property should be not be set either.</td>
<td>immediate</td>
</tr>
<tr>
<td>defaultCategory</td>
<td>String. Value selected as the default in the Category poplist when adding new attachments.</td>
<td></td>
</tr>
<tr>
<td>label</td>
<td>String. Specifies the title for the Attachment popup.</td>
<td>Attachment</td>
</tr>
<tr>
<td>rows</td>
<td>Number. Maximum number of rows to display in a single range of rows in attachment table. Some ranges may have fewer than the number specified here, i.e. the last range when there is an insufficient number of rows. To display all rows at once, set this attribute value to 0.</td>
<td>10</td>
</tr>
<tr>
<td>showCategory</td>
<td>Boolean: true / false. Controls show/hide of the Category column in the Attachments table.</td>
<td>True</td>
</tr>
<tr>
<td>auxiliaryOffset</td>
<td>Number. The offset shift factor that a carousel item will have relative to its nearest item towards the current carousel item in circular displayItems mode. This is the distance an auxiliary item will be moved from its nearest item towards the current item (the distance is the size of the next nearest item multiplied by this offset factor).</td>
<td></td>
</tr>
<tr>
<td>auxiliaryPopOut</td>
<td>String. Valid Values: off, hover; whether mouse movements within the auxiliary item area of the carousel in circular displayItems mode causes the auxiliary items to be scaled to match the current carousel item.</td>
<td>off</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>auxiliaryScale</td>
<td>Number. The size scaling factor that a carousel item will have relative to its nearest item towards the current carousel item in circular displayItems mode. A value of 1 means the auxiliary items will all be the same size. A value less than 1 means the auxiliary items will become smaller the further they are from the current item. A value greater than 1 means the auxiliary items will become larger the further they are from the current item. Note that regardless of item size, the current carousel item is the only item that is interactive; clicking on an auxiliary item will only spin the carousel.</td>
<td></td>
</tr>
<tr>
<td>controlArea</td>
<td>String. Valid Values: full, small, compact, none; specifies how the area where the user spins the carousel is presented. The “full” option lets a user spin through carousel items one at a time, jump to a specific item, or drag the slider thumb to continuously spin the carousel until the mouse is let go. The “small” option lets a user click the next and previous buttons to spin through carousel items one at a time. The “compact” option is similar to “small” but omits details such as the “x of y” progress information. The “none” option provides no control area at all.</td>
<td>full</td>
</tr>
<tr>
<td>displayItems</td>
<td>String. Valid Values: circular, oneByOne; the current carousel item will always be shown but you can use this attribute to specify how the auxiliary carousel items are displayed. Use “circular” if you want the auxiliary items displayed near the current item in a scaled down pattern with as many items shown as will fit. Use “oneByOne” if you only want the current item shown.</td>
<td>circular</td>
</tr>
<tr>
<td>emptyText</td>
<td>String. The text of an empty carousel. If the text is enclosed in an open and closing html tag, it will be formatted. The formatting behavior is similar to outputFormatted component. If it is not enclosed in an open and closing html tag, it will not be formatted.</td>
<td></td>
</tr>
<tr>
<td>ReadOnly</td>
<td>Boolean: true / false. True: disables all creation/update/delete functions in the Attachment component at runtime.</td>
<td>False</td>
</tr>
<tr>
<td>addAllowed</td>
<td>Boolean: true / false. Enables/disables the “Add” icon in the attachments table toolbar. True: enables the icon in the toolbar. False: disables the icon in the toolbar.</td>
<td>True</td>
</tr>
<tr>
<td>addEnabled</td>
<td>Boolean: true / false. Enables/disables the “Add” icon in the attachments table toolbar. True: shows the icon in the toolbar. False: hides the icon in the toolbar.</td>
<td>True</td>
</tr>
</tbody>
</table>
### Table 70-2 (Cont.) Attachments Carousel Component Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>updateAllowed</td>
<td>Boolean: <em>true / false</em>. Enables/disables the update icons in the attachments table toolbar. These include the &quot;Check In&quot;, &quot;Check Out&quot; and &quot;Cancel Check Out&quot; icons. This property also enables/disables the users' ability to update the Category, Title and Description of the Attachments. True: enables the icons and makes the Category, Title and Description columns editable for existing attachments. False: disables the update icons and makes the Category, Title and Description columns read-only for existing attachments.</td>
<td>True</td>
</tr>
<tr>
<td>updateEnabled</td>
<td>Boolean: <em>true / false</em>. Controls show/hide of the update icons in the attachments table toolbar. These include the &quot;Check In&quot;, &quot;Check Out&quot; and &quot;Cancel Check Out&quot; icons. True: shows the update icons in the toolbar. False: hides the update icons in the toolbar.</td>
<td>True</td>
</tr>
<tr>
<td>deleteAllowed</td>
<td>Boolean: <em>true / false</em>. Enables/disables the &quot;Delete&quot; icon in the attachments toolbar and inline for link and columnLink modes. True: Deleting attachments is allowed. True: enables the icon. False: disables the icon.</td>
<td>True</td>
</tr>
<tr>
<td>deleteEnabled</td>
<td>Boolean: <em>true / false</em>. Controls show/hide of the &quot;Delete&quot; icon in the attachments table toolbar. True: shows the icon in the toolbar. False: hides the icon in the toolbar preventing users from deleting attachments.</td>
<td>True</td>
</tr>
<tr>
<td>viewAllowed</td>
<td>Boolean: <em>true / false</em>. True: Viewing attachments is allowed. False: Links for viewing documents in Attachments tables are disabled.</td>
<td>True</td>
</tr>
<tr>
<td>viewEnabled</td>
<td>Boolean: <em>true / false</em>. True: shows the update icons in the toolbar. False: hides the update icons in the toolbar.</td>
<td>True</td>
</tr>
<tr>
<td>confirmDelete</td>
<td>Boolean: <em>true / false</em>. True: displays the Delete Attachment Confirmation page when the delete icon is used.</td>
<td>True</td>
</tr>
<tr>
<td>EnableOrdering</td>
<td>String. Allows the end-user to modify the display order of the attachments. This can be performed via keyboard commands or a reorder Attachments dialog launched from the Attachments Action menu.</td>
<td>False</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>carouselContentDelivery</td>
<td>Valid Values: immediate, lazy; whether data should be fetched when the component is rendered initially. When contentDelivery is &quot;immediate&quot;, data is fetched and inlined into the component chrome. If contentDelivery is &quot;lazy&quot;, data will be fetched and delivered to the client during a subsequent request.</td>
<td>lazy</td>
</tr>
<tr>
<td>fetchSize</td>
<td>Number. The number of rows in the data fetch block. Not supported on the following renderkits: org.apache.myfaces.trinidad.core</td>
<td>25</td>
</tr>
<tr>
<td>first</td>
<td>Number. The index of the first row in the currently range of rows. This index is zero-based. This attribute is used to control which range of rows to display to the user.</td>
<td>0</td>
</tr>
<tr>
<td>carouselImmediate</td>
<td>Boolean: true / false. Whether or not data validation - client-side or server-side - should take place when events are generated by this component. When immediate is true, the default ActionListener provided by the JavaServer Faces implementation should be executed during Apply Request Values phase of the request processing lifecycle, rather than waiting until the Invoke Application phase.</td>
<td>False</td>
</tr>
<tr>
<td>carouselRows</td>
<td>Number. The maximum number of rows to display in a single range of rows. Some ranges might have fewer than the number of rows specified by this attribute (eg: the last range might have an insufficient number of rows). To display all rows at once, set this attribute to 0. The default is 25.</td>
<td>25</td>
</tr>
<tr>
<td>RowBandingInterval</td>
<td>Number. The interval between which the row banding occurs. This value controls the display of the row banding in the table. For example, rowBandingInterval=1 would display alternately banded rows in the grid.</td>
<td></td>
</tr>
<tr>
<td>attachmentModel</td>
<td>CollectionModel</td>
<td></td>
</tr>
<tr>
<td>deleteMessage</td>
<td>String. Message to be displayed on Delete Attachment confirmation page.</td>
<td>Are you sure you want to delete the selected attachment(s)?</td>
</tr>
<tr>
<td>Description</td>
<td>String. Displays string as a tip when the user hovers the cursor over the Attachment component.</td>
<td></td>
</tr>
<tr>
<td>useResetActionListener</td>
<td>Boolean: true / false.</td>
<td>False</td>
</tr>
<tr>
<td>inlineStyle</td>
<td>String</td>
<td>width:950px</td>
</tr>
</tbody>
</table>
### Table 70-2  (Cont.) Attachments Carousel Component Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>carouselStyleClass</td>
<td>String. A Cascading Style Sheet (CSS) style class to use for this component. The style class can be defined in your string and skinning CSS file, for example, or you can use one of our public style classes, like &quot;AFInstructionText&quot;.</td>
<td>width:500.0px; height:300.0px;</td>
</tr>
<tr>
<td>carouselInlineStyle</td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. If the inlineStyle's CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td>width:500.0px; height:300.0px;</td>
</tr>
<tr>
<td>carouselItemInlineStyle</td>
<td>String. the CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. If the inlineStyle's CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td>width:200px;</td>
</tr>
<tr>
<td>UseResetActionListener</td>
<td>Boolean: true / false. True Activates the declarative af:resetActionListener which is associated with the OK and Cancel buttons for each popup in the Attachments component. The resetActionListener resets all of the values submitted to avoid the data in the form from being stale. For more information, see &quot;How to Use a Command Component to Reset Input Fields&quot; in Developing Web User Interfaces with Oracle ADF Faces.</td>
<td>False</td>
</tr>
<tr>
<td>fieldRenderOff</td>
<td>String. A space-separated list of fields that will not be rendered into the page. The fields are Attached By (attachedBy), Attached Date (attachedDate), Description, Shared, and Checked Out By (checkedOutBy).</td>
<td></td>
</tr>
<tr>
<td>carouselItemViewer</td>
<td>String: custom. When the property is set to custom, the Attachments Carousel component will use the customCarouselItem facet for the Carousel Item rather than the default Carousel Item.</td>
<td></td>
</tr>
</tbody>
</table>

A facet (placeholder) is provided on the Attachments Carousel component for product teams to develop their own Carousel Item. The facet customCarouselItem is provided. To enable the facet to be shown instead of the default Carousel Item set the property carouselItemViewer to custom.

The facet tableAppsTableSecondaryToolbar is provided to add toolbar buttons to the Attachments Table toolbar.
## 70.3 About Configuring the Attachment Views Components UI

The default behavior of the Attachment Views component can be changed by setting properties on the component using the Property Inspector.

### Table 70-3  Attachment Views Component Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>actionEntity</td>
<td>String. This must be an Attachment Entity Name. For use when showing the attachments for multiple entities in the Attachments table/popup. Setting this property indicates that update and delete actions will only be possible on this entity.</td>
<td></td>
</tr>
<tr>
<td>actionItemsRendered</td>
<td>String: true / false. Controls whether the Actions menu is rendered in the toolbar.</td>
<td>False</td>
</tr>
<tr>
<td>addAllowed</td>
<td>Boolean: true / false. Enables/disables the Add icon in the toolbar. True: enables the icon in the toolbar. False: disables the icon in the toolbar.</td>
<td>True</td>
</tr>
<tr>
<td>addEnabled</td>
<td>Boolean: true / false. True: shows the icon in the toolbar. False: hides the icon in the toolbar.</td>
<td>True</td>
</tr>
<tr>
<td>additionalListInlineActionItems</td>
<td>Boolean: true / false. True: shows the Inline Action Items facet for the List View. False: hides the facet.</td>
<td>False</td>
</tr>
<tr>
<td>additionalListInlineIconItems</td>
<td>Boolean: true / false. True: shows the Inline Icon Items facet for the List View. False: hides the facet.</td>
<td>False</td>
</tr>
<tr>
<td>additionalTableInlineActionItems</td>
<td>Boolean: true / false. True: shows the Inline Action Items facet for the Table View. False: hides the facet.</td>
<td>False</td>
</tr>
<tr>
<td>additionalTableInlineIconItems</td>
<td>Boolean: true / false. True: shows the Inline Icon Items facet for the Table View. False: hides the facet.</td>
<td>False</td>
</tr>
<tr>
<td>approvalEnabled</td>
<td>Boolean: true / false. True: Adds the Status field to the Detail pane.</td>
<td>False</td>
</tr>
<tr>
<td>attachmentModel</td>
<td>Collection Model. Required.</td>
<td></td>
</tr>
<tr>
<td>binding</td>
<td>Binding (EL only). An EL reference that will store the component instance on a bean. This can be used to give programmatic access to a component from a backing bean, or to move creation of the component to a backing bean.</td>
<td></td>
</tr>
<tr>
<td>confirmDelete</td>
<td>Boolean: true / false. True displays the Delete Attachment Confirmation dialog when the delete icon is used.</td>
<td>True</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultCategory</td>
<td>String. Value selected as the default in the Category poplist when adding new attachments.</td>
<td></td>
</tr>
<tr>
<td>deleteEnabled</td>
<td>Boolean: true / false. Controls show/hide of the Delete icon in the attachments table toolbar. True: shows the icon in the toolbar. False: hides the icon in the toolbar preventing users from deleting attachments.</td>
<td>True</td>
</tr>
<tr>
<td>deleteMessage</td>
<td>String. Message to be displayed on Delete Attachment confirmation page.</td>
<td></td>
</tr>
<tr>
<td>displayViews</td>
<td>String. A space-separated list of views to show. The views are all, table, and list. The default is all, which will show table and list views. It also automatically includes the planned grid and carousel views.</td>
<td>All</td>
</tr>
<tr>
<td>enablePrimaryCategory</td>
<td>String. Users can designate one attachment to be the primary from all of the attachments assigned to a category. Set to true to enable.</td>
<td>False</td>
</tr>
<tr>
<td>fieldRenderOff</td>
<td>String. A space-separated list of fields that will not be rendered into the page. The fields are Attached By (attachedBy), Attached Date (attachedDate), Description, Shared, and checked Out By (checkedOutBy).</td>
<td></td>
</tr>
<tr>
<td>gridItems</td>
<td>Number. Maximum number of cards to display for the grid view.</td>
<td>25</td>
</tr>
</tbody>
</table>
## Table 70-3  (Cont.) Attachment Views Component Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
</table>
| id           | String. The identifier for the component. Every component may be named by a component identifier that must conform to the following rules:
  • They must start with a letter (as defined by the Character.isLetter() method) or underscore ( _ ).
  • Subsequent characters must be letters (as defined by the Character.isLetter() method), digits as defined by the Character.isDigit() method, dashes (-), or underscores ( _ ). To minimize the size of responses generated by JavaServer Faces, it is recommended that component identifiers be as short as possible. If a component has been given an identifier, it must be unique in the namespace of the closest ancestor to that component that is a NamingContainer (if any). | Auto-generated               |
| label        | String. Specifies the title for the Attachment popup.                       | #{applcoreBundle.ATTACHMENT_LABEL} |
| listItems    | Number. Maximum number of rows to display for the List View.                | 10                           |
| listItemViewer | String.                                                                     | Default                      |
| orderingEnabled | String. Provides the end user with the option to control the order the attachment rows are displayed in the table. Set to true to enable. | False                        |
| readOnly     | Boolean: true / false. True disables all creation/update/delete functions in the Attachment component at runtime. | False                        |
| redlineKey   | String. Null, or a comma-separated concatenation of the Entity Name and PK1_VALUE through PK5_VALUEs. Empty tailing PK Values do not need to be included. Supplying a non-null value triggers a comparison of the Attachments for the current record and the record identified by the supplied key. |                              |
| rendered     | Boolean: true / false. True: Component is rendered in the page. False: the component is not rendered. | True                         |
| sortEnabled  | String: true / false. Controls the visibility of the Sort dropdown list in the toolbar. | False                        |
### Table 70-3 (Cont.) Attachment Views Component Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>sortFeaturesOff</td>
<td>String. A space-separated list of fields that will not be included in the Sort dropdown list. The fields are Title, Description, Category, Checked Out By (checkedOutBy), Shared, Last Updated By (lastUpdatedBy), Last Update Date (lastUpdateDate), Status, Size.</td>
<td></td>
</tr>
<tr>
<td>typesEnabled</td>
<td>String. A space-separated list of attachment types that are available for use in the Attachments Views component. The types are – all, file, text, and URL. The default &quot;all&quot; type automatically provides access to all current and any future Attachment types.</td>
<td>All</td>
</tr>
<tr>
<td>updateAllowed</td>
<td>Boolean: true / false. Enables/disables the users' ability to update the Category, Title and Description of the Attachments. True: Makes the Category, Title and Description columns editable for existing attachments. False: Makes the Category, Title and Description columns read-only for existing attachments.</td>
<td>True</td>
</tr>
<tr>
<td>updateCategoryList</td>
<td>String. Stores a comma-separated list of category names that will be used at runtime to populate the category LOV when a user adds new attachments or attempts to update the category of an existing attachment. This list must be a subset of the categories that are assigned to your entity.</td>
<td></td>
</tr>
<tr>
<td>updateEnabled</td>
<td>Boolean: true / false. Controls show/hide of the update icons in the toolbar. These include the &quot;Check In&quot;, &quot;Check Out&quot; and &quot;Cancel Check Out&quot; icons. True: shows the update icons in the toolbar. False: hides the update icons in the toolbar.</td>
<td>True</td>
</tr>
<tr>
<td>visible</td>
<td>Boolean: true / false. True: the component is displayed in the page. False: the component is hidden.</td>
<td>True</td>
</tr>
</tbody>
</table>

The Attachment Views component is a declarative component composed of a number of Oracle Application Development Framework components. The following are the principal components that make up the Attachment Views component:

- List
  - List Item
- Grid
  - Grid Item
• Carousel
  – Carousel Item

• Table
  – Column
    * Category
    * Title
    * Description
    * CheckedOutBy
    * Shared
    * LastUpdatedBy
    * Status
    * Size

Properties for each of these items that are not already used by the Attachments View to provide specific functionality have been exposed. To provide a unique name for common property names, a prefix has been added to the sub-components. This helps to differentiate the InlineStyle property on the Category column and the InlineStyle on the Title column from the InlineStyle of the table itself. For example, tableColumnCategoryInlineStyle, tableColumnTitleInlineStyle and tableInlineStyle are prepended with table.

For consistency, the label and visible properties for the Attachment fields have been consolidated into their own properties. Rather than having to override the label for Title in the table view, list view and Details pane, you can set the property just once and it will display across all of the occurrences.

The following table provides the full set of configuration properties, along with descriptions of what each property does, as found in the Oracle Application Development Framework tag documentation.

---

**Note:**

The properties for the Grid, Grid Item, Carousel and Carousel Item will be used when the Grid and Carousel views become available in the future.

---

**Table 70-4  Oracle Application Development Framework Configuration Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>listEmptyText</td>
<td>String. The text of an empty listview. If the text is enclosed in an open and closing HTML tag, it will be formatted. The formatting behavior is similar to outputFormatted component. If it is not enclosed in an open and closing html tag, it will not be formatted.</td>
<td>#{avlAttrs.listModel.viewable ? applcoreBundle.TABLE_E MPTY_TEXT_NO_ROWS_YET : applcoreBundle.TABLE_E MPTY_TEXT_ACCESS_DENIED}</td>
</tr>
</tbody>
</table>

Configure Attachments Component Properties 70-17
Table 70-4  (Cont.) Oracle Application Development Framework Configuration Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>listFetchSize</td>
<td>Integer. The number of items in the data fetch block. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>#{avlAttrs.listModel.rangeSize}</td>
</tr>
<tr>
<td>listInlineStyle</td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. Be aware that because of browser CSS precedence rules, CSS rendered on a DOM element takes precedence over external stylesheets like the skin file. Therefore skins will not be able to override what you set on this attribute. If the inlineStyle’s CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td></td>
</tr>
<tr>
<td>listStyleClass</td>
<td>String. A CSS style class to use for this component. The style class can be defined in your String. A CSS style class to use for this component. The style class can be defined in your .jspx page or in a skinning CSS file, for example, or you can use one of our public style classes, like &quot;AFInstructionText&quot;.</td>
<td></td>
</tr>
<tr>
<td>listItemInlineStyle</td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. Be aware that because of browser CSS precedence rules, CSS rendered on a DOM element takes precedence over external stylesheets like the skin file. Therefore skins will not be able to override what you set on this attribute. If the inlineStyle’s CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>listItemStyleClass</td>
<td>String. A CSS style class to use for this component. The style class can be defined in your String. A CSS style class to use for this component. The style class can be defined in your .jspx page or in a skinning CSS file, for example, or you can use one of our public style classes, like &quot;AFInstructionText&quot;.</td>
<td></td>
</tr>
<tr>
<td>gridEmptyText</td>
<td>String. The text of an empty ListView. If the text is enclosed in an open and closing html tag, it will be formatted. The formatting behavior is similar to outputFormatted component. If it is not enclosed in an open and closing HTML tag, it will not be formatted.</td>
<td></td>
</tr>
<tr>
<td>gridFetchSize</td>
<td>Integer. The number of items in the data fetch block Not supported on the following renderkits: org.apache.myfaces.trinidad.core</td>
<td>25</td>
</tr>
<tr>
<td>gridInlineStyle</td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. Be aware that because of browser CSS precedence rules, CSS rendered on a DOM element takes precedence over external stylesheets like the skin file. Therefore skins will not be able to override what you set on this attribute. If the inlineStyle's CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td></td>
</tr>
<tr>
<td>gridStyleClass</td>
<td>Integer. The number of items in the data fetch block Not supported on the following renderkits: org.apache.myfaces.trinidad.core</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>gridItemInlineStyle</td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. Be aware that because of browser CSS precedence rules, CSS rendered on a DOM element takes precedence over external stylesheets like the skin file. Therefore skins will not be able to override what you set on this attribute. If the inlineStyle's CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td></td>
</tr>
<tr>
<td>gridItemStyleClass</td>
<td>String. A CSS style class to use for this component. The style class can be defined in your .jspx page or in a skinning CSS file, for example, or you can use one of our public style classes, like &quot;AFInstructionText&quot;.</td>
<td></td>
</tr>
<tr>
<td>carouselAuxiliaryOffset</td>
<td>Number. The offset shift factor that a carousel item will have relative to its nearest item towards the current carousel item in circular displayItems mode. This is the distance an auxiliary item will be moved from its nearest item towards the current item (the distance is the size of the next nearest item multiplied by this offset factor).</td>
<td></td>
</tr>
<tr>
<td>carouselAuxiliaryPopOut</td>
<td>String. Whether mouse movements within the auxiliary item area of the carousel in circular displayItems mode causes the auxiliary items to be scaled to match the current carousel item. Valid values: off, hover.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>carouselAuxiliaryScale</td>
<td>Number. The size scaling factor that a carousel item will have relative to its nearest item towards the current carousel item in circular displayItems mode. A value of 1 means the auxiliary items will all be the same size. A value less than 1 means the auxiliary items will become smaller the further they are from the current item. A value greater than 1 means the auxiliary items will become larger the further they are from the current item. Note that regardless of item size, the current carousel item is the only item that is interactive; clicking on an auxiliary item will only spin the carousel.</td>
<td>lazy</td>
</tr>
<tr>
<td>carouselContentDelivery</td>
<td>String. Specifies if data should be fetched when the component is rendered initially. When contentDelivery is &quot;immediate&quot;, data is fetched and inlined into the component chrome. If contentDelivery is &quot;lazy&quot;, data will be fetched and delivered to the client during a subsequent request. Valid values: immediate, lazy</td>
<td></td>
</tr>
<tr>
<td>carouselControlArea</td>
<td>String. Specifies how the area where the user spins the carousel is presented. The &quot;full&quot; option lets a user spin through carousel items one at a time, jump to a specific item, or drag the slider thumb to continuously spin the carousel until the mouse is let go. The &quot;small&quot; option lets a user click the Next and Previous to spin through carousel items one at a time. The &quot;compact&quot; option is similar to &quot;small&quot; but omits details such as the &quot;x of y&quot; progress information. The &quot;none&quot; option provides no control area at all. Valid Values: full, small, compact, none.</td>
<td>full</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>carouselDisplayItems</td>
<td>String. The current carousel item will always be shown, but you can use this attribute to specify how the auxiliary carousel items are displayed. Use &quot;circular&quot; if you want the auxiliary items displayed near the current item in a scaled down pattern with as many items shown as will fit. Use &quot;oneByOne&quot; if you only want the current item shown. Valid Values: circular, oneByOne.</td>
<td>circular</td>
</tr>
<tr>
<td>carouselEmptyText</td>
<td>String. The text of an empty carousel. If the text is enclosed in an open and closing HTML tag, it will be formatted. The formatting behavior is similar to outputFormatted component. If it is not enclosed in an open and closing HTML tag, it will not be formatted.</td>
<td></td>
</tr>
<tr>
<td>carouselFetchSize</td>
<td>Integer. The number of rows in the data fetch block Not supported on the following renderkits: org.apache.myfaces.trinidad.core</td>
<td>25</td>
</tr>
<tr>
<td>carouselHalign</td>
<td>String. Determines the horizontal alignment of the carousel items. Valid values: start, center, end.</td>
<td>center</td>
</tr>
<tr>
<td>carouselImmediate</td>
<td>Boolean. Specifies if data validation – client-side or server-side – should take place when events are generated by this component. When immediate is true, the default ActionListener provided by the JavaServer Faces implementation should be executed during Apply Request Values phase of the request processing lifecycle, rather than waiting until the Invoke Application phase.</td>
<td>False</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>carouselInlineStyle</td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. Be aware that because of browser CSS precedence rules, CSS rendered on a DOM element takes precedence over external stylesheets like the skin file. Therefore skins will not be able to override what you set on this attribute. If the inlineStyle's CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td></td>
</tr>
<tr>
<td>carouselOrientation</td>
<td>String. Horizontal (the default) for the images being displayed along an x-axis or vertical for y-axis. Valid values: horizontal, vertical.</td>
<td>horizontal</td>
</tr>
<tr>
<td>carouselStyleClass</td>
<td>String. A CSS style class to use for this component. The style class can be defined in your String. A CSS style class to use for this component. The style class can be defined in your .jspx page or in a skinning CSS file, for example, or you can use one of our public style classes, like &quot;AFInstructionText&quot;.</td>
<td></td>
</tr>
<tr>
<td>carouselValign</td>
<td>String. Determines the vertical alignment of the carousel items. Valid values: middle, top, bottom.</td>
<td>middle</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>carouselItemInlineStyle</td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. Be aware that because of browser CSS precedence rules, CSS rendered on a DOM element takes precedence over external stylesheets like the skin file. Therefore skins will not be able to override what you set on this attribute. If the inlineStyle's CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td></td>
</tr>
<tr>
<td>carouselItemStyleClass</td>
<td>String. A CSS style class to use for this component. The style class can be defined in your string. A CSS style class to use for this component. The style class can be defined in your .jspx page or in a skinning CSS file, for example, or you can use one of our public style classes, like “AFInstructionText”.</td>
<td></td>
</tr>
<tr>
<td>carouselItemText</td>
<td>String. The Text of the carousel item.</td>
<td></td>
</tr>
<tr>
<td>tableAutoHeightRows</td>
<td>Number. Sets the maximum number of rows that the table height will automatically adjust to depending on the amount of data in the table.</td>
<td>10</td>
</tr>
<tr>
<td>tableColumnBandingInterval</td>
<td>Number. The interval between which the column banding occurs. This value controls the display of the column banding in the table. For example, columnBandingInterval=1 would display alternately banded columns in the grid.</td>
<td></td>
</tr>
<tr>
<td>tableColumnResizing</td>
<td>String. Setting this value to &quot;disabled&quot; turns off user column resizing. The default is &quot;enabled&quot;. Valid values: enabled, disabled</td>
<td>enabled</td>
</tr>
</tbody>
</table>
Table 70-4  (Cont.) Oracle Application Development Framework Configuration Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableColumnStretching</td>
<td>Indicates what kind of stretching to apply to the columns. For optimal performance, use the &quot;none&quot; (default) option. To stretch the last column to fill up any unused space inside of the viewport, use the &quot;last&quot; option. To automatically insert an empty blank column and have it stretch (so the row background colors will span the entire width of the table), use the &quot;blank&quot; option. To stretch a specific leaf (non-group) column to fill up any unused space inside of the viewport, use the option where you specify &quot;column:&quot; followed by the ID of the column that you want to have stretched, for example, &quot;column:myColId&quot;. To stretch more than one column, use the &quot;multiple&quot; option. When using the &quot;multiple&quot; option, you can set the widths in the columns (non-group) to be percentages. See the column documentation for how percentages are handled. Note that row headers and frozen columns will not be stretched because doing so could easily leave the user unable to access the scrollable data body of the table. Not supported on the following renderkits: org.apache.myfaces.trinidad.core</td>
<td>column:colDesc</td>
</tr>
<tr>
<td>tableContentDelivery</td>
<td>String. Use this property only with the AutoHeightRows property. If AutoHeightRows is not set, this property also should be not be set.</td>
<td>Immediate</td>
</tr>
<tr>
<td>tableDisableColumnReordering</td>
<td>Boolean. Specifies if column reordering is disabled in the table. Column reordering is on by default. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>False</td>
</tr>
<tr>
<td>tableEmptyText</td>
<td>String. The text of an empty table. If the text is enclosed in an open and closing HTML tag, it will be formatted. The formatting behavior is similar to outputFormatted component. If it is not enclosed in an open and closing HTML tag, it will not be formatted.</td>
<td>#{avtAttrs.tblModel.viewable ? applcoreBundle.TABLEEMPTY_TEXT_NO_ROWS_YET : applcoreBundle.TABLEEMPTY_TEXT_ACCESS_DENIED}</td>
</tr>
</tbody>
</table>
### Table 70-4  (Cont.) Oracle Application Development Framework Configuration Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableEmptyText</td>
<td>Integer. The number of rows in the data fetch block. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.#{avtAttrs.tblModel.rangeSize}</td>
<td>#{avtAttrs.tblModel.rangeSize}</td>
</tr>
<tr>
<td>tableFreezeDirection</td>
<td>String. Determines if the column-freezing zone is from the specified column to the start or the end of the table row. Valid values: start, end.</td>
<td>start</td>
</tr>
<tr>
<td>tableHorizontalGridVisible</td>
<td>Boolean. Specifies if the Horizontal GridLines are to be drawn.</td>
<td>True</td>
</tr>
<tr>
<td>tableInlineStyle</td>
<td>String. Inline style that is applied to the table in the table view.</td>
<td>width:99%</td>
</tr>
<tr>
<td>tableMaxClientRows</td>
<td>Integer. Specifies the maximum number of rows to retain on the client when scrollPolicy=&quot;loadMore&quot;.</td>
<td>500</td>
</tr>
<tr>
<td>tableRowBandingInterval</td>
<td>Number. The interval at which row banding occurs. This value controls the display of the row banding in the table. For example, rowBandingInterval=1 would display alternately banded rows in the grid.</td>
<td></td>
</tr>
<tr>
<td>tableRows</td>
<td>Integer. Determines the number of rows to show in email mode, print mode, or other in situations where table scrolling is unavailable. This property will not have an affect outside of these situations.</td>
<td>10</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>tableShortDesc</td>
<td>String. The short description of the component. The shortDesc text may be used in two different ways, depending on the component. For components with images, the shortDesc is often used to render an HTML &quot;Alt&quot; attribute for the image. shortDesc is also commonly used to render an HTML title attribute, which is used by user agents to display &quot;tooltip&quot; help text. In this case, the behavior of the tooltip is controlled by the user agent. For example, Firefox 2 truncates long tooltips. For form components, the shortDesc is displayed in a note window. For components that support the helpTopicId attribute and are not using the shortDesc as image Alt text, it is recommended that helpTopicId is used instead of shortDesc because it is more flexible and provides more accessible descriptive text than the use of the title attribute.</td>
<td>#{applcoreBundle.AA_TABLE_SUMMARY}</td>
</tr>
<tr>
<td>tableStyleClass</td>
<td>String. A CSS style class to use for this component. The style class can be defined in your .jspx page or in a skinning CSS file, for example, or you can use one of our public style classes, like &quot;AFInstructionText&quot;.</td>
<td>AFStretchWidth</td>
</tr>
<tr>
<td>tableSummary</td>
<td>String. The summary of the table’s purpose and structure for user agents rendering to non-visual media.</td>
<td>#{applcoreBundle.AA_TABLE_SUMMARY}</td>
</tr>
<tr>
<td>tableVerticalGridVisible</td>
<td>Boolean. Specifies if the vertical GridLines are to be drawn.</td>
<td>True</td>
</tr>
<tr>
<td>tableWidth</td>
<td>String. The width of the component. Default value is 300 pixels. Do not use 100% since it does not account for borders and padding. Use styleClass = &quot;AFStretchWidth&quot; in order to achieve 100%.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 70-4  (Cont.) Oracle Application Development Framework Configuration Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableColumnCategoryAlign</td>
<td>String. The alignment for this column. The values &quot;start&quot;, &quot;end&quot;, and &quot;center&quot; are used for left-justified, right-justified, and center-justified respectively in LTR display. Use &quot;left&quot; or &quot;right&quot; when left-justified or right-justified cells are needed irrespective of the LTR or RTL display. The default value is null, which implies that it is skin dependent and may vary for the row header column vs data column. Valid values: start, end, center, left, right</td>
</tr>
<tr>
<td>tableColumnCategoryDisplayIndex</td>
<td>Integer. The display order index of the column. Columns can be rearranged and are displayed in the table based on the displayIndex. Columns are sorted based on the displayIndex property; columns without displayIndex are displayed at the end, in the order in which they appear. The displayIndex attribute is honored only for top-level columns, since it is not possible to rearrange a child column outside of the parent column. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
</tr>
<tr>
<td>tableColumnCategoryFooterClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells will cause misalignment of the table cells.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tableColumnCategoryFrozen</td>
<td>Boolean. Specifies if the column is frozen. In the table, the frozen column is locked with the header and cannot be scrolled with the rest of the columns. The frozen attribute is honored only on the top-level column, since it is not possible to freeze a child column by itself without its parent being frozen. If the table has a detailStamp for its rows, column freezing is turned off. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
</tr>
<tr>
<td>tableColumnCategoryHeaderClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells cause misalignment of the table cells.</td>
</tr>
<tr>
<td>tableColumnCategoryHeaderNoWrap</td>
<td>Boolean. Specifies if the column header should be allowed to wrap. Not supported on the following renderkits: oracle.adf.rich.</td>
</tr>
<tr>
<td>tableColumnCategoryInlineStyle</td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. Be aware that because of browser CSS precedence rules, CSS rendered on a DOM element takes precedence over external stylesheets like the skin file. Therefore skins will not be able to override what you set on this attribute. If the inlineStyle's CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
</tr>
</tbody>
</table>
### Table 70-4  (Cont.) Oracle Application Development Framework Configuration Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableColumnCategoryMinimumWidth</td>
<td>String. The minimum number of pixels that the column can sized. When a user attempts to resize the column, this minimum width will be enforced. Also, when a column is flexible, it will also never be stretched to be a size smaller than this minimum width. If a pixel width is defined and if the minimum width is larger, the minimum width will become the smaller of the two values. By default, the minimum width is 12 pixels. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>12</td>
</tr>
<tr>
<td>tableColumnCategoryNoWrap</td>
<td>Boolean. Specifies if whitespace wrapping should be allowed in this column.</td>
<td>True</td>
</tr>
<tr>
<td>tableColumnCategoryStyleClass</td>
<td>String. A CSS style class to use for this component. The style class can be defined in your .jspx page or in a skinning CSS file, for example, or you can use one of our public style classes, like &quot;AFInstructionText&quot;.</td>
<td></td>
</tr>
<tr>
<td>tableColumnCategoryWidth</td>
<td>String. The width of the column. The default column width is 100px. There is no auto-sizing for columns. Set the width attribute to ensure the column is wide enough to accommodate the width of the contents. When the &quot;multiple&quot; option is used in the table that contains the column, the width can be set to a percentage. Non-group column widths as percentages will be honored. When widths using percentages exist and the table columnStretching is set to &quot;multiple&quot;, the percentages are used to determine a normalized ratio. For example, if two columns have their widths set to &quot;100%&quot;, they will each be normalized to take up .5 times the stretchable space. Stretchable space is defined as the space after pixel widths are taken into account. Minimum widths are honored by the stretching.</td>
<td>100</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>tableColumnTitleAlign</td>
<td>String. The alignment for this column. The values &quot;start&quot;, &quot;end&quot;, and &quot;center&quot; are used for left-justified, right-justified, and center-justified respectively in LTR display. Use &quot;left&quot; or &quot;right&quot; when left-justified or right-justified cells are needed irrespective of the LTR or RTL display. The default value is null, which implies that it is skin dependent and may vary for the row header column vs data column. Valid values: start, end, center, left, right</td>
<td></td>
</tr>
<tr>
<td>tableColumnTitleDisplayIndex</td>
<td>Integer. The display order index of the column. Columns can be rearranged and they are displayed in the table based on the displayIndex. Columns are sorted based on the displayIndex property, columns without displayIndex are displayed at the end, in the order in which they appear. The displayIndex attribute is honored only for top-level columns, since it is not possible to rearrange a child column outside of the parent column. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>-1</td>
</tr>
<tr>
<td>tableColumnTitleFooterClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells will cause misalignment of the table cells.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>tableColumnTitleFrozen</td>
<td>Boolean. Specifies if the column is frozen. In the table, the frozen column is locked with the header and cannot be scrolled with the rest of the columns. The frozen attribute is honored only on the top-level column, since it is not possible to freeze a child column by itself without its parent being frozen. If the table has a detailStamp for its rows, column freezing is turned off. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>False</td>
</tr>
<tr>
<td>tableColumnTitleHeaderClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells will cause misalignment of the table cells.</td>
<td></td>
</tr>
<tr>
<td>tableColumnTitleHeaderNoWrap</td>
<td>Boolean. Specifies if the column header should be allowed to wrap Not supported on the following renderkits: oracle.adf.rich.</td>
<td>False</td>
</tr>
<tr>
<td>tableColumnTitleInlineStyle</td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. Be aware that because of browser CSS precedence rules, CSS rendered on a DOM element takes precedence over external stylesheets like the skin file. Therefore skins will not be able to override what you set on this attribute. If the inlineStyle’s CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>tableColumnTitleMinimumWidth</td>
<td>String. The minimum number of pixels that the column can sized. When a user attempts to resize the column, this minimum width will be enforced. Also, when a column is flexible, it will also never be stretched to be a size smaller than this minimum width. If a pixel width is defined and if the minimum width is larger, the minimum width will become the smaller of the two values. By default, the minimum width is 12 pixels. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>12</td>
</tr>
<tr>
<td>tableColumnTitleNoWrap</td>
<td>Boolean. Specifies if whitespace wrapping should be allowed in this column.</td>
<td>True</td>
</tr>
<tr>
<td>tableColumnTitleStyleClass</td>
<td>String. A CSS style class to use for this component. The style class can be defined in your .jspx page or in a skinning CSS file, for example, or you can use one of our public style classes, like &quot;AFInstructionText&quot;.</td>
<td></td>
</tr>
<tr>
<td>tableColumnTitleWidth</td>
<td>String. The width of the column. The default column width is 100px. There is no auto-sizing for columns. Set the width attribute to ensure the column is wide enough to accommodate the width of the contents. When the &quot;multiple&quot; option is used in the table that contains the column, the width can be set to a percentage. Non-group column widths as percentages will be honored. When widths using percentages exist and the table columnStretching is set to &quot;multiple&quot;, the percentages are used to determine a normalized ratio. For example, if two columns have their widths set to &quot;100%&quot;, they will each be normalized to take up .5 times the stretchable space. Stretchable space is defined as the space after pixel widths are taken into account. Minimum widths are honored by the stretching.</td>
<td>100</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>tableColumnDescriptionAlign</td>
<td>String. The alignment for this column. The values &quot;start&quot;, &quot;end&quot;, and &quot;center&quot; are used for left-justified, right-justified, and center-justified respectively in LTR display. Use &quot;left&quot; or &quot;right&quot; when left-justified or right-justified cells are needed irrespective of the LTR or RTL display. The default value is null, which implies that it is skin dependent and may vary for the row header column vs data column. Valid values: start, end, center, left, right.</td>
<td></td>
</tr>
<tr>
<td>tableColumnDescriptionDisplayIndex</td>
<td>Integer. The display order index of the column. Columns can be rearranged and they are displayed in the table based on the displayIndex. Columns are sorted based on the displayIndex property, columns without displayIndex are displayed at the end, in the order in which they appear. The displayIndex attribute is honored only for top-level columns, since it is not possible to rearrange a child column outside of the parent column. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>-1</td>
</tr>
<tr>
<td>tableColumnDescriptionFooterClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells cause misalignment of the table cells.</td>
<td></td>
</tr>
<tr>
<td>tableColumnDescriptionFrozen</td>
<td>Boolean. Specifies if the column is frozen. In the table, the frozen column is locked with the header and cannot be scrolled with the rest of the columns. The frozen attribute is honored only on the top-level column, since it is not possible to freeze a child column by itself without its parent being frozen. If the table has a detailStamp for its rows, column freezing is turned off. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>False</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td>tableColumnDescriptionHeaderClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells cause misalignment of the table cells.</td>
<td></td>
</tr>
<tr>
<td>tableColumnDescriptionHeaderNoWrap</td>
<td>Boolean. Specifies if the column header should be allowed to wrap</td>
<td>False</td>
</tr>
<tr>
<td>tableColumnDescriptionInlineStyle</td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. Be aware that because of browser CSS precedence rules, CSS rendered on a DOM element takes precedence over external stylesheets like the skin file. Therefore skins will not be able to override what you set on this attribute. If the inlineStyle’s CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td></td>
</tr>
<tr>
<td>tableColumnDescriptionMinimumWidth</td>
<td>String. The minimum number of pixels that the column can sized. When a user attempts to resize the column, this minimum width will be enforced. Also, when a column is flexible, it will also never be stretched to be a size smaller than this minimum width. If a pixel width is defined and if the minimum width is larger, the minimum width will become the smaller of the two values. By default, the minimum width is 12 pixels. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>12</td>
</tr>
<tr>
<td>tableColumnDescriptionNoWrap</td>
<td>Boolean. Specifies if whitespace wrapping should be allowed in this column.</td>
<td>True</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>tableColumnDescriptionStyleClass</td>
<td>String. A CSS style class to use for this component. The style class can be defined in your .jspx page or in a skinning CSS file, for example, or you can use one of our public style classes, like &quot;AFInstructionText&quot;.</td>
<td></td>
</tr>
<tr>
<td>tableColumnDescriptionWidth</td>
<td>String. The width of the column. The default column width is 100px. There is no auto-sizing for columns. Set the width attribute to ensure the column is wide enough to accommodate the width of the contents. When the “multiple” option is used in the table that contains the column, the width can be set to a percentage. Non-group column widths as percentages will be honored. When widths using percentages exist and the table columnStretching is set to “multiple”, the percentages are used to determine a normalized ratio. For example, if two columns have their widths set to “100%”, they will each be normalized to take up .5 times the stretchable space. Stretchable space is defined as the space after pixel widths are taken into account. Minimum widths are honored by the stretching.</td>
<td>100</td>
</tr>
<tr>
<td>tableColumnCheckedOutByAlign</td>
<td>String. The alignment for this column. The values “start”, “end”, and “center” are used for left-justified, right-justified, and center-justified respectively in LTR display. Use “left” or “right” when left-justified or right-justified cells are needed irrespective of the LTR or RTL display. The default value is null, which implies that it is skin dependent and may vary for the row header column vs data column. Valid values: start, end, center, left, right</td>
<td></td>
</tr>
</tbody>
</table>
### Table 70-4 (Cont.) Oracle Application Development Framework Configuration Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableColumnCheckedOutByDisplayIndex</td>
<td>Integer. The display order index of the column. Columns can be rearranged and are displayed in the table based on the displayIndex. Columns are sorted based on the displayIndex property; columns without displayIndex are displayed at the end, in the order in which they appear. The displayIndex attribute is honored only for top-level columns, since it is not possible to rearrange a child column outside of the parent column. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>-1</td>
</tr>
<tr>
<td>tableColumnCheckedOutByFooterClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells cause misalignment of the table cells.</td>
<td></td>
</tr>
<tr>
<td>tableColumnCheckedOutByFrozen</td>
<td>Boolean. Specifies if the column is frozen. In the table, the frozen column is locked with the header and cannot be scrolled with the rest of the columns. The frozen attribute is honored only on the top-level column, since it is not possible to freeze a child column by itself without its parent being frozen. If the table has a detailStamp for its rows, column freezing is turned off. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>False</td>
</tr>
<tr>
<td>tableColumnCheckedOutByHeaderClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells cause misalignment of the table cells.</td>
<td></td>
</tr>
<tr>
<td>tableColumnCheckedOutByHeaderNoWrap</td>
<td>Boolean. Specifies if the column header should be allowed to wrap. Not supported on the following renderkits: oracle.adf.rich.</td>
<td>False</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>tableColumnCheckedOutByInlineStyle</td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. Be aware that because of browser CSS precedence rules, CSS rendered on a DOM element takes precedence over external stylesheets like the skin file. Therefore skins will not be able to override what you set on this attribute. If the inlineStyle's CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td></td>
</tr>
<tr>
<td>tableColumnCheckedOutByMinimumWidth</td>
<td>String. The minimum number of pixels that the column can sized. When a user attempts to resize the column, this minimum width will be enforced. Also, when a column is flexible, it will also never be stretched to be a size smaller than this minimum width. If a pixel width is defined and if the minimum width is larger, the minimum width will become the smaller of the two values. By default, the minimum width is 12 pixels. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>12</td>
</tr>
<tr>
<td>tableColumnCheckedOutByNoWrap</td>
<td>Boolean. Specifies if whitespace wrapping should be allowed in this column.</td>
<td>True</td>
</tr>
<tr>
<td>tableColumnCheckedOutByStyleClass</td>
<td>String. A CSS style class to use for this component. The style class can be defined in your String. A CSS style class to use for this component. The style class can be defined in your .jspx page or in a skinning CSS file, for example, or you can use one of our public style classes, like &quot;AFInstructionText&quot;.</td>
<td></td>
</tr>
</tbody>
</table>
Table 70-4  (Cont.) Oracle Application Development Framework Configuration Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableColumnCheckedOutByWidth</td>
<td>String. The width of the column. The default column width is 100px. There is no auto-sizing for columns. Set the width attribute to ensure the column is wide enough to accommodate the width of the contents. When the “multiple” option is used in the table that contains the column, the width can be set to a percentage. Non-group column widths as percentages will be honored. When widths using percentages exist and the table columnStretching is set to &quot;multiple&quot;, the percentages are used to determine a normalized ratio. For example, if two columns have their widths set to &quot;100%&quot;, they will each be normalized to take up .5 times the stretchable space. Stretchable space is defined as the space after pixel widths are taken into account. Minimum widths are honored by the stretching.</td>
<td>100</td>
</tr>
<tr>
<td>tableColumnSharedAlign</td>
<td>String. The alignment for this column. The values “start”, “end”, and “center” are used for left-justified, right-justified, and center-justified respectively in LTR display. Use “left” or “right” when left-justified or right-justified cells are needed irrespective of the LTR or RTL display. The default value is null, which implies that it is skin dependent and may vary for the row header column vs data column. Valid values: start, end, center, left, right</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>tableColumnSharedDisplayIndex</td>
<td>Integer. The display order index of the column. Columns can be rearranged and are displayed in the table based on the displayIndex. Columns are sorted based on the displayIndex property; columns without displayIndex are displayed at the end, in the order in which they appear. The displayIndex attribute is honored only for top-level columns, since it is not possible to rearrange a child column outside of the parent column. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>-1</td>
</tr>
<tr>
<td>tableColumnSharedFooterClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells cause misalignment of the table cells.</td>
<td></td>
</tr>
<tr>
<td>tableColumnSharedFrozen</td>
<td>Boolean. Specifies if the column is frozen. In the table, the frozen column is locked with the header and cannot be scrolled with the rest of the columns. The frozen attribute is honored only on the top-level column, since it is not possible to freeze a child column by itself without its parent being frozen. If the table has a detailStamp for its rows, column freezing is turned off. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>False</td>
</tr>
<tr>
<td>tableColumnSharedHeaderClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells cause misalignment of the table cells.</td>
<td></td>
</tr>
<tr>
<td>tableColumnSharedHeaderNoWrap</td>
<td>Boolean. Specifies if the column header should be allowed to wrap.</td>
<td>False</td>
</tr>
</tbody>
</table>

About Configuring the Attachment Views Components UI
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableColumnSharedInlineStyle</td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. Be aware that because of browser CSS precedence rules, CSS rendered on a DOM element takes precedence over external stylesheets like the skin file. Therefore skins will not be able to override what you set on this attribute. If the inlineStyle's CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td></td>
</tr>
<tr>
<td>tableColumnSharedMinimumWidth</td>
<td>String. The minimum number of pixels that the column can sized. When a user attempts to resize the column, this minimum width will be enforced. Also, when a column is flexible, it will also never be stretched to be a size smaller than this minimum width. If a pixel width is defined and if the minimum width is larger, the minimum width will become the smaller of the two values. By default, the minimum width is 12 pixels. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>12</td>
</tr>
<tr>
<td>tableColumnSharedNoWrap</td>
<td>Boolean. Specifies if whitespace wrapping should be allowed in this column.</td>
<td>True</td>
</tr>
<tr>
<td>tableColumnSharedStyleClass</td>
<td>String. A CSS style class to use for this component. The style class can be defined in your String. A CSS style class to use for this component. The style class can be defined in your .jspx page or in a skinning CSS file, for example, or you can use one of our public style classes, like &quot;AFInstructionText&quot;.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td><code>tableColumnSharedWidth</code></td>
<td>String. The width of the column. The default column width is 100px. There is no auto-sizing for columns. Set the width attribute to ensure the column is wide enough to accommodate the width of the contents. When the “multiple” option is used in the table that contains the column, the width can be set to a percentage. Non-group column widths as percentages will be honored. When widths using percentages exist and the table columnStretching is set to “multiple”, the percentages are used to determine a normalized ratio. For example, if two columns have their widths set to “100%”, they will each be normalized to take up .5 times the stretchable space. Stretchable space is defined as the space after pixel widths are taken into account. Minimum widths are honored by the stretching.</td>
<td>100</td>
</tr>
<tr>
<td><code>tableColumnLastUpdatedByAlign</code></td>
<td>String. The alignment for this column. The values “start”, “end”, and “center” are used for left-justified, right-justified, and center-justified respectively in LTR display. Use “left” or “right” when left-justified or right-justified cells are needed irrespective of the LTR or RTL display. The default value is null, which implies that it is skin dependent and may vary for the row header column vs data column. Valid values: start, end, center, left, right</td>
<td></td>
</tr>
</tbody>
</table>
**Table 70-4  (Cont.) Oracle Application Development Framework Configuration Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableColumnLastUpdatedByDisplayIndex</td>
<td>Integer. The display order index of the column. Columns can be rearranged and are displayed in the table based on the displayIndex. Columns are sorted based on the displayIndex property; columns without displayIndex are displayed at the end, in the order in which they appear. The displayIndex attribute is honored only for top-level columns, since it is not possible to rearrange a child column outside of the parent column. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>-1</td>
</tr>
<tr>
<td>tableColumnLastUpdatedByFooterClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells will cause misalignment of the table cells.</td>
<td></td>
</tr>
<tr>
<td>tableColumnLastUpdatedByFrozen</td>
<td>Boolean. Specifies if the column is frozen. In the table, the frozen column is locked with the header and cannot be scrolled with the rest of the columns. The frozen attribute is honored only on the top-level column, since it is not possible to freeze a child column by itself without its parent being frozen. If the table has a detailStamp for its rows, column freezing is turned off. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>False</td>
</tr>
<tr>
<td>tableColumnLastUpdatedByHeaderClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells will cause misalignment of the table cells.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>tableColumnLastUpdatedByHeaderNoWrap</td>
<td>Boolean. Specifies if the column header should be allowed to wrap. Not supported on the following renderkits: oracle.adf.rich.</td>
<td></td>
</tr>
<tr>
<td>tableColumnLastUpdatedByInlineStyle</td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. Be aware that because of browser CSS precedence rules, CSS rendered on a DOM element takes precedence over external stylesheets like the skin file. Therefore skins will not be able to override what you set on this attribute. If the inlineStyle's CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td></td>
</tr>
<tr>
<td>tableColumnLastUpdatedByMinimumWidth</td>
<td>String. The minimum number of pixels that the column can sized. When a user attempts to resize the column, this minimum width will be enforced. Also, when a column is flexible, it will also never be stretched to be a size smaller than this minimum width. If a pixel width is defined and if the minimum width is larger, the minimum width will become the smaller of the two values. By default, the minimum width is 12 pixels. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>12</td>
</tr>
<tr>
<td>tableColumnLastUpdatedByNoWrap</td>
<td>Boolean. Specifies if whitespace wrapping should be allowed in this column.</td>
<td>True</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>tableColumnLastUpdatedByStyleClass</td>
<td>String. A CSS style class to use for this component. The style class can be defined in your String. A CSS style class to use for this component. The style class can be defined in your .jspx page or in a skinning CSS file, for example, or you can use one of our public style classes, like &quot;AFInstructionText&quot;.</td>
<td></td>
</tr>
<tr>
<td>tableColumnLastUpdatedByWidth</td>
<td>String. The width of the column. The default column width is 100px. There is no auto-sizing for columns. Set the width attribute to ensure the column is wide enough to accommodate the width of the contents. When the &quot;multiple&quot; option is used in the table that contains the column, the width can be set to a percentage. Non-group column widths as percentages will be honored. When widths using percentages exist and the table columnStretching is set to &quot;multiple&quot;, the percentages are used to determine a normalized ratio. For example, if two columns have their widths set to &quot;100%&quot;, they will each be normalized to take up .5 times the stretchable space. Stretchable space is defined as the space after pixel widths are taken into account. Minimum widths are honored by the stretching.</td>
<td>100</td>
</tr>
<tr>
<td>tableColumnLastUpdatedDateAlign</td>
<td>String. The alignment for this column. The values &quot;start&quot;, &quot;end&quot;, and &quot;center&quot; are used for left-justified, right-justified, and center-justified respectively in LTR display. Use &quot;left&quot; or &quot;right&quot; when left-justified or right-justified cells are needed irrespective of the LTR or RTL display. The default value is null, which implies that it is skin dependent and may vary for the row header column vs data column. Valid values: start, end, center, left, right</td>
<td></td>
</tr>
</tbody>
</table>
### Table 70-4  (Cont.) Oracle Application Development Framework Configuration Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableColumnLastUpdatedDateDisplayIndex</td>
<td>Integer. The display order index of the column. Columns can be rearranged and are displayed in the table based on the displayIndex. Columns are sorted based on the displayIndex property; columns without displayIndex are displayed at the end, in the order in which they appear. The displayIndex attribute is honored only for top-level columns, since it is not possible to rearrange a child column outside of the parent column. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>-1</td>
</tr>
<tr>
<td>tableColumnLastUpdatedDateFooterClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells will cause misalignment of the table cells.</td>
<td></td>
</tr>
<tr>
<td>tableColumnLastUpdatedDateFrozen</td>
<td>Boolean. Specifies if the column is frozen. In the table, the frozen column is locked with the header and cannot be scrolled with the rest of the columns. The frozen attribute is honored only on the top-level column, since it is not possible to freeze a child column by itself without its parent being frozen. If the table has a detailStamp for its rows, column freezing is turned off. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>False</td>
</tr>
<tr>
<td>tableColumnLastUpdatedDateHeaderClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells will cause misalignment of the table cells.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>tableColumnLastUpdatedDateHeaderNoWrap</td>
<td>Boolean. Specifies if the column header should be allowed to wrap. Not supported on the following renderkits: oracle.adf.rich.</td>
<td>False</td>
</tr>
<tr>
<td>tableColumnLastUpdatedDateInlineStyle</td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. Be aware that because of browser CSS precedence rules, CSS rendered on a DOM element takes precedence over external stylesheets like the skin file. Therefore skins will not be able to override what you set on this attribute. If the inlineStyle’s CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td></td>
</tr>
<tr>
<td>tableColumnLastUpdatedDateMinimumWidth</td>
<td>String. The minimum number of pixels that the column can sized. When a user attempts to resize the column, this minimum width will be enforced. Also, when a column is flexible, it will also never be stretched to be a size smaller than this minimum width. If a pixel width is defined and if the minimum width is larger, the minimum width will become the smaller of the two values. By default, the minimum width is 12 pixels. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>12</td>
</tr>
<tr>
<td>tableColumnLastUpdatedDateNoWrap</td>
<td>Boolean. Specifies if whitespace wrapping should be allowed in this column.</td>
<td>True</td>
</tr>
<tr>
<td>tableColumnLastUpdatedDateStyleClass</td>
<td>String. A CSS style class to use for this component. The style class can be defined in your .jspx page or in a skinning CSS file, for example, or you can use one of our public style classes, like &quot;AFInstructionText&quot;.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>tableColumnLastUpdatedDateWidth</td>
<td>String. The width of the column. The default column width is 100px. There is no auto-sizing for columns. Set the width attribute to ensure the column is wide enough to accommodate the width of the contents. When the “multiple” option is used in the table that contains the column, the width can be set to a percentage. Non-group column widths as percentages will be honored. When widths using percentages exist and the table columnStretching is set to “multiple”, the percentages are used to determine a normalized ratio. For example, if two columns have their widths set to “100%”, they will each be normalized to take up .5 times the stretchable space. Stretchable space is defined as the space after pixel widths are taken into account. Minimum widths are honored by the stretching.</td>
<td>100</td>
</tr>
<tr>
<td>tableColumnStatusAlign</td>
<td>String. The alignment for this column. The values “start”, “end”, and “center” are used for left-justified, right-justified, and center-justified respectively in LTR display. Use “left” or “right” when left-justified or right-justified cells are needed irrespective of the LTR or RTL display. The default value is null, which implies that it is skin dependent and may vary for the row header column vs data column. Valid values: start, end, center, left, right</td>
<td>-1</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>tableColumnStatusDisplayIndex</td>
<td>Integer. The display order index of the column. Columns can be rearranged and are displayed in the table based on the displayIndex. Columns are sorted based on the displayIndex property; columns without displayIndex are displayed at the end, in the order in which they appear. The displayIndex attribute is honored only for top-level columns, since it is not possible to rearrange a child column outside of the parent column. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td></td>
</tr>
<tr>
<td>tableColumnStatusFooterClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells will cause misalignment of the table cells.</td>
<td></td>
</tr>
<tr>
<td>tableColumnStatusFrozen</td>
<td>Boolean. Specifies if the column is frozen. In the table, the frozen column is locked with the header and cannot be scrolled with the rest of the columns. The frozen attribute is honored only on the top-level column, since it is not possible to freeze a child column by itself without its parent being frozen. If the table has a detailStamp for its rows, column freezing is turned off. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>False</td>
</tr>
<tr>
<td>tableColumnStatusHeaderClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells will cause misalignment of the table cells.</td>
<td></td>
</tr>
</tbody>
</table>
Table 70-4  (Cont.) Oracle Application Development Framework Configuration Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableColumnStatusHeaderNoWrap</td>
<td>Boolean. Specifies if the column header should be allowed to wrap. Not supported on the following renderkits: oracle.adf.rich.</td>
<td>False</td>
</tr>
<tr>
<td>tableColumnStatusInlineStyle</td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. Be aware that because of browser CSS precedence rules, CSS rendered on a DOM element takes precedence over external stylesheets like the skin file. Therefore skins will not be able to override what you set on this attribute. If the inlineStyle's CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td></td>
</tr>
<tr>
<td>tableColumnStatusMinimumWidth</td>
<td>String. The minimum number of pixels that the column can sized. When a user attempts to resize the column, this minimum width will be enforced. Also, when a column is flexible, it will also never be stretched to be a size smaller than this minimum width. If a pixel width is defined and if the minimum width is larger, the minimum width will become the smaller of the two values. By default, the minimum width is 12 pixels. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>12</td>
</tr>
<tr>
<td>tableColumnStatusNoWrap</td>
<td>Boolean. Specifies if whitespace wrapping should be allowed in this column.</td>
<td>True</td>
</tr>
<tr>
<td>tableColumnStatusStyleClass</td>
<td>String. A CSS style class to use for this component. The style class can be defined in your .jspx page or in a skinning CSS file, for example, or you can use one of our public style classes, like &quot;AFInstructionText&quot;.</td>
<td></td>
</tr>
</tbody>
</table>

About Configuring the Attachment Views Components UI
<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableColumnStatusWidth</td>
<td>String. The width of the column. The default column width is 100px. There is no auto-sizing for columns. Set the width attribute to ensure the column is wide enough to accommodate the width of the contents. When the &quot;multiple&quot; option is used in the table that contains the column, the width can be set to a percentage. Non-group column widths as percentages will be honored. When widths using percentages exist and the table columnStretching is set to &quot;multiple&quot;, the percentages are used to determine a normalized ratio. For example, if two columns have their widths set to &quot;100%&quot;, they will each be normalized to take up .5 times the stretchable space. Stretchable space is defined as the space after pixel widths are taken into account. Minimum widths are honored by the stretching.</td>
<td>100</td>
</tr>
<tr>
<td>tableColumnSizeAlign</td>
<td>String. The alignment for this column. The values &quot;start&quot;, &quot;end&quot;, and &quot;center&quot; are used for left-justified, right-justified, and center-justified respectively in LTR display. Use &quot;left&quot; or &quot;right&quot; when left-justified or right-justified cells are needed irrespective of the LTR or RTL display. The default value is null, which implies that it is skin dependent and may vary for the row header column vs data column. Valid values: start, end, center, left, right</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>tableColumnSizeDisplayIndex</td>
<td>Integer. The display order index of the column. Columns can be rearranged and are displayed in the table based on the displayIndex. Columns are sorted based on the displayIndex property; columns without displayIndex are displayed at the end, in the order in which they appear. The displayIndex attribute is honored only for top-level columns, since it is not possible to rearrange a child column outside of the parent column. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>-1</td>
</tr>
<tr>
<td>tableColumnSizeFooterClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells will cause misalignment of the table cells.</td>
<td></td>
</tr>
<tr>
<td>tableColumnSizeFrozen</td>
<td>Boolean. Specifies if the column is frozen. In the table, the frozen column is locked with the header and cannot be scrolled with the rest of the columns. The frozen attribute is honored only on the top-level column, since it is not possible to freeze a child column by itself without its parent being frozen. If the table has a detailStamp for its rows, column freezing is turned off. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>False</td>
</tr>
<tr>
<td>tableColumnSizeHeaderClass</td>
<td>String. A CSS style class to use for the column footer. Use the headerClass, footerClass, and styleClass attributes on the column with caution. Changing the horizontal padding/borders of the header, footer and data cells will cause misalignment of the table cells.</td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td><code>tableColumnSizeHeaderNoWrap</code></td>
<td>Boolean. Specifies if the column header should be allowed to wrap. Not supported on the following renderkits: oracle.adf.rich.</td>
<td>False</td>
</tr>
<tr>
<td><code>tableColumnSizeInlineStyle</code></td>
<td>String. The CSS styles to use for this component. This is intended for basic style changes. The inlineStyle is a set of CSS styles that are applied to the root DOM element of the component. Be aware that because of browser CSS precedence rules, CSS rendered on a DOM element takes precedence over external stylesheets like the skin file. Therefore skins will not be able to override what you set on this attribute. If the inlineStyle's CSS properties do not affect the DOM element you want affected, then you will have to create a skin and use the skinning keys which are meant to target particular DOM elements, like ::label or ::icon-style.</td>
<td></td>
</tr>
<tr>
<td><code>tableColumnSizeMinimumWidth</code></td>
<td>String. The minimum number of pixels that the column can sized. When a user attempts to resize the column, this minimum width will be enforced. Also, when a column is flexible, it will also never be stretched to be a size smaller than this minimum width. If a pixel width is defined and if the minimum width is larger, the minimum width will become the smaller of the two values. By default, the minimum width is 12 pixels. Not supported on the following renderkits: org.apache.myfaces.trinidad.core.</td>
<td>12</td>
</tr>
<tr>
<td><code>tableColumnSizeNoWrap</code></td>
<td>Boolean. Specifies if whitespace wrapping should be allowed in this column.</td>
<td>True</td>
</tr>
<tr>
<td><code>tableColumnSizeStyleClass</code></td>
<td>String. A CSS style class to use for this component. The style class can be defined in your .jspx page or in a skinning CSS file, for example, or you can use one of our public style classes, like &quot;AFInstructionText&quot;.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 70-4  (Cont.) Oracle Application Development Framework Configuration Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>tableColumnSizeWidth</td>
<td>String. The width of the column. The default column width is 100px. There is no auto-sizing for columns. Set the width attribute to ensure the column is wide enough to accommodate the width of the contents. When the &quot;multiple&quot; option is used in the table that contains the column, the width can be set to a percentage. Non-group column widths as percentages will be honored. When widths using percentages exist and the table columnStretching is set to &quot;multiple&quot;, the percentages are used to determine a normalized ratio. For example, if two columns have their widths set to &quot;100%&quot;, they will each be normalized to take up .5 times the stretchable space. Stretchable space is defined as the space after pixel widths are taken into account. Minimum widths are honored by the stretching.</td>
<td>100</td>
</tr>
<tr>
<td>attributeCategoryLabel</td>
<td>String. Overrides the default label for all occurrences in the Attachments View component.</td>
<td></td>
</tr>
<tr>
<td>attributeCategoryVisible</td>
<td>Boolean: true / false. True: the field is displayed in the view. False: the field is hidden.</td>
<td>false</td>
</tr>
<tr>
<td>attributeTitleLabel</td>
<td>String. Overrides the default label for all occurrences in the Attachments View component.</td>
<td></td>
</tr>
<tr>
<td>attributeDescriptionLabel</td>
<td>String. Overrides the default label for all occurrences in the Attachments View component.</td>
<td></td>
</tr>
<tr>
<td>attributeDescriptionVisible</td>
<td>Boolean: true / false. True: the field is displayed in the view. False: the field is hidden.</td>
<td>true</td>
</tr>
<tr>
<td>attributeCheckedOutByLabel</td>
<td>String. Overrides the default label for all occurrences in the Attachments View component.</td>
<td></td>
</tr>
<tr>
<td>attributeCheckedOutByVisible</td>
<td>Boolean: true / false. True: the field is displayed in the view. False: the field is hidden.</td>
<td>false</td>
</tr>
<tr>
<td>attributeSharedLabel</td>
<td>String. Overrides the default label for all occurrences in the Attachments View component.</td>
<td></td>
</tr>
</tbody>
</table>
**Table 70-4 (Cont.) Oracle Application Development Framework Configuration Properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributeSharedVisible</td>
<td>Boolean: true / false. True: the field is displayed in the view. False: the field is hidden.</td>
<td>False</td>
</tr>
<tr>
<td>attributeLastUpdatedByLabel</td>
<td>String. Overrides the default label for all occurrences in the Attachments View component.</td>
<td></td>
</tr>
<tr>
<td>attributeLastUpdatedByVisible</td>
<td>Boolean: true / false. True: the field is displayed in the view. False: the field is hidden.</td>
<td>False</td>
</tr>
<tr>
<td>attributeLastUpdatedDateLabel</td>
<td>String. Overrides the default label for all occurrences in the Attachments View component.</td>
<td></td>
</tr>
<tr>
<td>attributeLastUpdatedDateVisible</td>
<td>Boolean: true / false. True: the field is displayed in the view. False: the field is hidden.</td>
<td>False</td>
</tr>
<tr>
<td>attributeStatusLabel</td>
<td>String. Overrides the default label for all occurrences in the Attachments View component.</td>
<td></td>
</tr>
<tr>
<td>attributeStatusVisible</td>
<td>Boolean: true / false. True: the field is displayed in the view. False: the field is hidden.</td>
<td>False</td>
</tr>
<tr>
<td>attributeSizeLabel</td>
<td>String. Overrides the default label for all occurrences in the Attachments View component.</td>
<td></td>
</tr>
<tr>
<td>attributeSizeVisible</td>
<td>Boolean: true / false. True: the field is displayed in the view. False: the field is hidden.</td>
<td>False</td>
</tr>
</tbody>
</table>

### 70.4 About Configuring the Attachment Link Component UI

The default behavior of the Attachment Link Component can be changed by setting properties on the component using the Property Inspector. The majority of the properties available are only used to configure how the Attachment Views component appears in the popup. See Configuring the Attachment Views Component UI (http://aseng-wiki.us.oracle.com/asengwiki/display/ATG/Release+10+Uptake#Release10Uptake-ConfiguringtheAttachmentViewsComponentUI) for more information on these properties. The following table documents the properties that directly affect the Attachment Link component.

**Table 70-5 Attachment Link Component Properties.**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachmentModel</td>
<td>Collection Model. Required.</td>
<td>No default value</td>
</tr>
</tbody>
</table>
### Table 70-5 (Cont.) Attachment Link Component Properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String. The identifier for the component. Every component may be named by a component identifier that must conform to the following rules: They must start with a letter (as defined by the Character.isLetter() method) or underscore ( _ ). Subsequent characters must be letters (as defined by the Character.isLetter() method), digits as defined by the Character.isDigit() method, dashes ( - ), or underscores ( _ ). To minimize the size of responses generated by JavaServer Faces, it is recommended that component identifiers be as short as possible. If a component has been given an identifier, it must be unique in the namespace of the closest ancestor to that component that is a NamingContainer (if any).</td>
<td>Auto-generated</td>
</tr>
<tr>
<td>linkMode</td>
<td>String: Classic, Abridged. Required. Specifies how the component should display. Classic will display with information about the last attachment, and action icons. Abridged will display as a single icon.</td>
<td>Classic</td>
</tr>
<tr>
<td>numAttachmentsDisplayed</td>
<td>Number. Maximum number of attachments displayed as links in the hover popup when mode is set to link or columnLink.</td>
<td>3</td>
</tr>
<tr>
<td>linkDeleteAllowed</td>
<td>String. Enables the display of the delete icon. This setting will not override the deleteAllowed or readOnly properties, which will also affect the display of the delete icon.</td>
<td>true</td>
</tr>
<tr>
<td>savepointListener</td>
<td>AttachmentSavepointListener.</td>
<td>No default value</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
<td>Default Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>skipValidationOnPopup</td>
<td>String; default, true, false, skipDataControls, validateCurrentRows. Controls whether the model validation is skipped for the binding container, while the Manage Attachments popup is displayed. The default value does not change the skip validation configuration. Set to false to perform a normal validation. Set to true to skip all validation. Set to skipDataControls to skip only data control validation. Set to validateCurrentRows to validate only the rows modified in the last request.</td>
<td>default</td>
</tr>
</tbody>
</table>
**Glossary**

**all-segment secondary usage**
A type of secondary usage in which the secondary table has all of the key flexfield segment columns that are present in the combinations table.

**application role**
A role specific to applications and stored in the policy store.

**Application Technology Group (ATG)**
Also known as ApplCore for "Applications Core."
A layer of runtime and design time features that deliver functionality targeted specifically for Oracle Fusion Applications.

**autosuggest**
Suggestions that automatically appear for a search field, even before you finish typing your search term. You can select any of the suggestions to run your search.

**business component**
One of a set of cooperating components that are used by ADF Business Components to implement a business service.

**business object**
A resource in an enterprise database, such as an invoice or purchase order.

**CCID**
Code combination ID, the common attribute for key flexfields. Every key flexfield combinations table must have a CCID column, the values of which identify each data row.

**child view object**
A view object that is nested within a master view object.
**code combination ID (CCID)**

See CCID.

**code-combination reference page**

Pages whose underlying entity objects contain a foreign key reference to the combinations table.

**combination maintenance page**

A page whose underlying entity objects use the combinations table itself.

**consumer (of a flexfield)**

The person who incorporates a flexfield into their application, which is typically different from the producer's application. The consumer typically stores the CCID on a transaction table, and works with the structural and seed data and the business components that have been configured by the key flexfield producer.

**context**

A group of attributes. Each context is part of a flexfield, and is comprised of a set of context-sensitive segments that store a particular type of related information.

**context attribute**

The flexfield base view object attribute that contains the context discriminator value.

**context-sensitive segment**

A flexfield segment that is a member of a context.

**custom validation callout**

Callout procedure that is used to enforce custom validation logic for new code combinations beyond what has been defined for cross-validation rules.

**customer flexfield**

A flexfield created for use by the customer.

**cross-validation rule**

A rule that is composed of a condition filter and a validation filter and that specifies when and how to validate a key flexfield code combination.

**data security**

The control of access to data. Data security controls what action a user can take against which data.
**database schema**

A named collection of objects, such as tables, views, clusters, procedures, packages, attributes, object classes, and their corresponding matching rules, which are associated with a particular user.

**derived segment**

A segment whose values automatically change to reflect new reference values.

**descriptive flexfield**

A type of flexfield used to give additional attributes to a data model. Allows customers to add custom attributes to entities, to define how the attributes are validated, and display properties for the attributes. These attributes do not necessarily have anything to do with each other and are not treated together as a combination. A descriptive flexfield can only support a set amount of segments.

**developer (of a flexfield)**

The role to be used if you are incorporating the flexfield into an application.

**developer flexfield**

A flexfield created to support functionality that has been built into the application.

**discriminator**

A common attribute amongst multiple view rows. The discriminator determines which view row type should be used.

**dynamic column**

A method of accessing a flexfield in Excel. Using a dynamic column will enable you to pick flexfield segment values. You can also enter values directly into the segment fields.

**dynamic combination insertion**

The act of end users entering values on an application page that constitute new code combinations (even if the end users are not authorized to perform maintenance tasks directly).

**EAR**

An Enterprise Archive file. A Java EE archive file that is used in deploying applications on a Java EE application server. Framework applications are deployed using both a generic EAR file, which contains the application and the respective runtime customization, and a targeted EAR file, which contains only the application for deployment to the application server.
Enterprise Archive (EAR)

See EAR.

etitlement

Grants of access to functions and data. Oracle Fusion Middleware term for privilege.

descriptive flexfield

A type of flexfield that is similar to a descriptive flexfield, but does not have a fixed number of segments, allows attributes to be grouped, allows entities to inherit segments from their parents, and supports one-to-many relationships between entity and extended attribute rows.

entity object

An object that represents a row in a database table and that simplifies modifying its data by handling all data manipulation language operations for you. Entity objects are ADF Business Components that provide the mapping to underlying data structures.

flexfield

An "expandable" data field that is divided into segments. Flexfields enable you to configure your applications to meet your business needs without having to perform custom development.

flexfield parameter

A declared public variable which can be used to designate which attributes of eligible entity objects that are related to the flexfield can be used to pass external reference data to flexfield segments. These entity objects could, in turn, take their values from column values, constant values, session attributes, and so forth.

form layout

A typical label/prompt and either view-only data or widget (text field, choice list, and so on) that allows a user to enter values.

function security

The control of access to a page or a specific widget or functionality within a page. Function security controls what a user can do.

global

The type of segment label that is used if you want the implementer to tie it to all segments.

globalsearch

The search in the global area that lets you search across many business objects.
global segment
A segment of custom attributes that apply to all entity rows.

hierarchical categories
A feature of extensible flexfields. Extensible flexfields can be configured to enable categories, which can be used to dynamically display different sets of logical pages and contexts based upon a runtime differentiator. The categories can be structured in a hierarchical manner and the children categories inherit all the contexts and logical pages that are configured for the parent categories.

implementor (of a flexfield)
An individual who sets up all or part of a flexfield-enabled application for deployment. Implementors typically work for or on behalf of customers to install, configure, or administer their applications. In the case of developer flexfields that have been created to support functionality that has been built into the application, the developer also plays the role of implementor.

intelligent key
A key composed of business-related values as opposed to an arbitrarily generated sequence.

key flexfield
A type of flexfield in which the segments define a key, or code, that uniquely identifies an object such as an account, an asset, a part, or a job.

maintenance mode
A mode of the key flexfield user interface which allows you to use a code combination maintenance page to manage key flexfield code combinations, including the ability to enter new code combinations and update existing code combinations for a flexfield.

MAR
A Metadata Archive file. A compressed archive of selected metadata used to deploy metadata content to the MDS Repository.

master view object
A view object that has other view objects nested within it.

Metadata Archive (MAR)
See MAR.
owner (of a flexfield)

The developer (or development team) who determines that a particular flexfield is needed or would be useful within a particular Oracle Fusion application, and makes a flexfield of the appropriate design available.

primary key attribute

The attribute of a base view object which links the flexfield view object to the application view object.

primary table

The application table that was used to first register the flexfield. It is the owner of the flexfield.

privilege

A grant or entitlement of access to functions and data.

producer (of a flexfield)

The developer who determines that a particular flexfield is needed or would be useful within a particular application, and makes available a flexfield of the appropriate design. With key flexfields, the producer's product owns the combinations table for that flexfield.

readonly

A boolean property that indicates whether users can modify the value.

rendered

A boolean property that indicates whether the value is visible on the application page.

required

A boolean property that indicates whether the field must have a value.

required (segment label)

The type of segment label that is used if you want the implementors to tie it at least to one segment.

role

Controls access to application functions and data.

secondary table

For descriptive flexfields, a table, other than the primary table, that contains the same set of extension columns as the primary table and that enables the primary table's
**descriptive flexfield** to be reused for that table. For key flexfields, a product table other than the combinations table that contains one or all of the key **flexfield** segments.

**secondary usage**

For descriptive flexfields, the reuse of a **flexfield** on a table other than the **primary table**.

For key flexfields, a type of usage in which there is no direct relationship between the product table and the combinations table.

See also **single-segment secondary usage** and **all-segment secondary usage**.

**security reference implementation**

Predefined function and **data security** in Oracle Fusion Applications, including **role** based access control, and policies that protect functions, data, and segregation of duties. The reference implementation supports identity management, access provisioning, and security enforcement across the tools, data transformations, access methods, and the information life cycle of an enterprise.

**segment label**

A label which identifies the purpose of a particular segment in a key **flexfield**.

**SIN**

Structure instance number. A segment that acts as the **discriminator** attribute for key flexfields.

**single-segment secondary usage**

A type of **secondary usage** in which the **secondary table** has only one **key flexfield** segment column.

**static column**

A method of accessing a **flexfield** in Excel. Used when the flexfield is exposed in an ADF Table component, is **context** sensitive, and the context changes from row to row. A static column should also be used if you do not want flexfield segments to occupy too much space in the worksheet.

**structure code**

A string you provide that individually identifies a structure.

**Structure instance number (SIN)**

See **SIN**.

**tester (of a flexfield)**

The **role** to be used if you are planning to test or share your **flexfield**.
suggestion group
Category of suggestions that appear in the autosuggest for the global search.

transient attribute
An attribute whose value is not stored in a database, and therefore holds a value only for the life of the object. Transient attributes are typically used to display values that are calculated (for example, using Java or Groovy).

unique
The type of segment label that is used if you want the implementer to tie it to at most one segment of the flexfield.

value set
A list of valid values used to specify the validation rules for a flexfield segment.

view accessor
An ADF Business Components object that points from an entity object attribute (or view object) to a destination view object or shared view instance in the same application workspace. The view accessor returns a row set that by default contains all the rows from the destination view object.

WAR
A Web Application Archive file. This file is used in deploying applications on a Java EE application server. WAR files encapsulate in a single module all of the components necessary to run an application. WAR files typically contain an application’s servlet, JSP, and JSF JSP components.

Web Application Archive (WAR)
See WAR.
Index

A

ADF Security
  best practices
    for task flows, 52-7
application server connection, 55-14

deploying ADF applications (continued)
  from JDeveloper, 55-2
  to EAR file, 55-2
    using application server connections, 55-14

B

best practices
  task flows
    securing, 52-7

D

deploying ADF applications

E

EAR files for deploying ADF applications, 55-2

T

task flows
  best practices
    for securing, 52-7