Oracle® Fusion Applications

Developer's Guide

11*g* Release 1 (11.1.2)

E15524-02

October 2011

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.

ORACLE®

Oracle Fusion Applications Developer's Guide 11*g* Release 1 (11.1.2)

E15524-02

Primary Author: Shelly Butcher, Richard Gugeler, Karen Ram, Karen Summerly, Chris Kutler, Ralph Gordon, Peter Jew

Contents

3 Setting Up Your JDeveloper Workspace and Projects

6 Defining Defaulting and Derivation Logic

7 Defining and Using Message Dictionary Messages

8 Managing Reference Data with SetIDs

9 Using Fusion Middleware Extensions for Oracle Applications Base Classes

10 Implementing Lookups

15 Implementing UIs in JDeveloper with Application Tables, Trees and Tree Tables

16 Implementing Applications Panels, Master-Detail, Hover, and Dialog Details

17 Implementing Skinning

18 Implementing Attachments

19 Organizing Hierarchical Data with Tree Structures

20 Working with Localization Formatting

23 Using Key Flexfields

24 Using Extensible Flexfields

25 Testing and Deploying Flexfields

Part V Using Oracle Enterprise Crawl and Search Framework

26 Getting Started with Oracle Enterprise Crawl and Search Framework

27 Creating Searchable Objects

28 Configuring ECSF Security

29 Validating and Testing Search Metadata

30 Deploying and Crawling Searchable Objects

31 Advanced Topics for ECSF

Part VI Common Service Use Cases and Design Patterns

32 Initiating a SOA Composite from an Oracle ADF Web Application

33 Initiating a SOA Composite from a PL/SQL Stored Procedure

34 Orchestrating ADF Business Components Services

35 Manipulating Back-End Data from a SOA Composite

43 Managing Tasks Programmatically

44 Implementing an Oracle ADF Task Flow for a Human Task

47 Implementing Application User Sessions

48 Implementing Oracle Fusion Data Security

49 Implementing Function Security

50 Securing Web Services Use Cases

51 Securing End-to-End Portlet Applications

Part VIII Advanced Topics

52 Running and Deploying Applications on Oracle WebLogic Server

53 Creating Repository Connections

54 Defining Profiles

55 Initializing Oracle Fusion Application Data Using the Seed Data Loader

56 Using the Database Schema Deployment Framework

57 Improving Performance

58 Debugging Oracle ADF and Oracle SOA Suite

59 Designing and Securing View Objects for Oracle Business Intelligence Applications

A Working with the Application Taxonomy

B ECSF Command Line Administration Utility

Preface

Welcome to the *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.

Audience

This document is intended for Oracle Applications Developers and assumes familiarity with Java and SQL.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

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

Related Documents

For more information, see the following documents in the Oracle 11g Fusion Middleware documentation set:

- Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework
- Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework
- Oracle Fusion Middleware Mobile Browser Developer's Guide for Oracle Application Development Framework
- Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework

- Oracle Fusion Middleware Installation Guide for Oracle Application Development Framework Skin Editor
- Oracle Fusion Middleware Skin Editor User's Guide for Oracle Application Development Framework
- 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 Modeling and Implementation Guide for Oracle Business Process Management
- Oracle Fusion Middleware Business Process Composer User's Guide for Oracle Business Process Management
- Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator
- Oracle Fusion Middleware Connectivity and Knowledge Modules Guide for Oracle Data Integrator
- Oracle Fusion Middleware Knowledge Module Developer's Guide for Oracle Data Integrator
- Oracle Fusion Middleware Developer's Guide for Oracle Imaging and Process Management
- Oracle Fusion Middleware Application Developer's Guide for Oracle Identity Management
- Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite
- Oracle Fusion Middleware User's Guide for Oracle Business Rules
- Oracle Fusion Middleware Language Reference Guide for Oracle Business Rules
- Oracle Fusion Middleware User's Guide for Technology Adapters
- Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring
- Oracle Fusion Middleware Developer's Guide for Oracle WebCenter
- Oracle Fusion Middleware User's Guide for Oracle WebCenter Spaces
- Oracle WebLogic Communication Services Developer's Guide
- Oracle Fusion Middleware Application Security Guide
- Oracle Fusion Applications Security Guide
- Oracle Fusion Applications Security Hardening and Best Practices Guide

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in This Guide for Release 11.1.2

This chapter provides information about what is new in the Oracle Fusion Applications Developer's Guide11g Release 1 (11.1.2) since Release 1 (11.1.1.5) was released in August 2011.

Documentation Changes for Release 11.1.2

For Release 11.1.2, this guide has been updated in several ways. The following table lists the sections that have been added or changed.

For changes made to Oracle JDeveloper and Oracle Application Development Framework (Oracle ADF) for this release, see the What's New page on the Oracle Technology Network at http://www.oracle.com/technetwork/developer-tools/jdev/documentation/index.html.

Sections	Changes Made
Chapter 2 Setting Up Your Development Environment	
Section 2.2.1.3, "Installing JDeveloper"	Section revised with the current procedure to install JDeveloper.
Section 2.2.1.5, "Setting Up the JDeveloper-based Development Environment"	Section updated with current instructions for adding the extensions bundle.
Section 2.2.2, "How to Use the Oracle Fusion Domain Wizard"	Section updated with current images and descriptions of wizard parameters.
Chapter 7 Defining Defaulting and Derivation Logic	
Section 6.3, "Using Oracle ADF Validators and Convertor Hints"	Section revised to include information about how to override an Oracle ADF validator or converter message with new text.
Chapter 16 Implementing Applications Panels, Master-Detail, Hover, and Dialog Details	
Section 16.1.2.1, "Adding Applications Panels Using the Applications Panel Wizard"	Section revised with expanded information for using the customSaveDropButton facet in description of Figure 16-7 Select Page-Level Buttons Dialog.
Chapter 23 Using Extensible Flexfields	
Section 24.9, "Customizing the Extensible Flexfield Runtime User Interface Modeler"	Section added that describes how to customize the extensible flexfield runtime user interface modeler.
Chapter 25 Testing and Deploying Flexfields	

Sections	Changes Made
Section 25.3, "Using the WLST Flexfield Commands"	Section revised to include new `deleteFlexPatchingLabels` WLST command for inquiring about or deleting MDS flexfield patching labels.
Chapter 23 Implementing Attachments	
Section 18.2.4, "How to Delete the Business Object"	Section added describing how to delete a business object and its Attachments.
Chapter 27 Creating Searchable Objects	
Section 27.4.4.4, "Defining A Facet to Use A Child View Object Attribute"	Section added describing new feature of being able to search on a child View Object.
Section 27.2.10.1, "Checking for Stored Attribute Conflicts"	Section added describing how ECSF now checks for stored attribute conflicts when defining a search.
Section 27.2.6.1, "Making View Object Attributes Searchable"	Section revised with expanded information of Crawl Date Column in Table 27-1 Searchable Attribute Properties to explain that ECSF will pick the most recent Crawl Date column value to use as the LastModifiedDate value.

Part I

Getting Started Building Your Oracle Fusion Applications

This part of the Developer's Guide discusses how to set up and configure your development environment to build your Oracle Fusion Applications using Oracle JDeveloper.

Getting Started with Oracle Fusion Applications describes how to design and build your Oracle Fusion Applications using the Oracle standards and guidelines.

Setting Up Your Development Environment describes how to configure and test your 11g 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 Oracle Integrated WebLogic Server and Oracle Standalone WebLogic Server, and the basic steps for setting up your service-oriented architecture (SOA) development environment.

Setting Up Your JDeveloper Workspace and Projects describes how to create an application so that the system automatically creates your Model and user interface projects. Also included are instructions about how to set up your projects including manually adding the *Applications Core* library to the data model project and the *Applications Core Tag* library to the user interface project.

This part contains the following chapters:

1

Getting 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 the following sections:

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 Fusion Developer's Guide for Oracle Application Development Framework*.

  For more information about ADF Faces rich client components, see the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

- **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 Desktop Integration Developer's Guide for Oracle Application Development Framework*.

- **ADF Mobile:**

  This technology is used to create interfaces that can be accessed through browsers in mobile devices.

  For more information about ADF Mobile, see the *Oracle Fusion Middleware Mobile Browser Developer's Guide for Oracle Application Development Framework*.

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 *Oracle Database Administrator's Guide*.

- Oracle Essbase: 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. OracleAS BPEL Process Manager is used to assemble these web services together to provide end-to-end functionality.

For more information about SOA, see the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

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).

In addition to customization, WebCenter provides many other services. For more information about WebCenter technologies, see the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

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 OracleAS 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 *Oracle Fusion Applications Developer's Guide for Oracle Enterprise Scheduler*.

- **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 Chapter 26, "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 *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.

- **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 Developer's Guide for Oracle Data Integrator*.

1.2 Using Oracle ADF Functional Patterns and Best Practices

The *Oracle ADF Functional Patterns and Best Practices* web site contains documents that describe and demonstrate functional patterns and best practices for specific tasks in development when utilizing the Oracle Application Development Framework (Oracle ADF) within JDeveloper. New functional patterns and best practices will appear on a regular basis. Also remember to check JDeveloper's online help and search the Web for more information that might be published on blogs.

The functional patterns and best practices discussed on the web site include:

- Oracle User Interface Shell
- Accessibility Global Link
- Unsaved Changes
- ADF Region Interaction
- Enabling and Disabling a UI Component.

2

Setting Up Your Development Environment

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:

2.1 Introduction to the Development Environment

Note: This chapter assumes that you are using a 64-bit operating system.

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
- Personal environment

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

2.1.1 Shared Environment

The shared environment normally will be set up by an administrator. This environment is completely provisioned and set up on a machine that is more powerful than the normal developer's machine, which often is a laptop. It is called the shared environment because developers will share its resources.

When it is provisioned, the shared environment contains:

- **Database**

  When provisioning installs and configures the database, it makes sure that all the necessary schemas are created in it. Provisioning also ensures that the Functional Setup is run, the taxonomy tables are populated, and the FlexFields are defined. The same database also contains the ApplicationDB schema that contains the data that developers see when working with the applications.

- **Oracle Middleware Home**

  Provisioning creates a complete Middleware home while creating the shared environment. Middleware home contains individual Oracle homes for product families, Oracle Business Intelligence, Oracle Fusion applications, WebLogic Server, and so on. Middleware home contains the exploded EAR (archive) directories of the deployed applications. The provisioning processes update and modify the **connections.xml** and the **adf-config.xml** files in the exploded EAR directories of all the deployed applications to point to the correct host, port, and endpoint details, based on where the database and the WebLogic Server domains have been provisioned in the shared environment.

- **WebLogic Server domains**

  The shared environment has one or more Weblogic Server domains running with Oracle Fusion applications deployed. Each domain will have one AdminServer and at most three ManagedServers. One of the applications may use web services from another application. As a result, the **connections.xml** will have references to the endpoint defined in other app. These domains will be useful in performing system tests.

- **Identity Store and Policy Store**

  The Identity Store and the Policy Store are not provisioned by the provisioning process. The administrator follows the Identity Management documentation and processes to set up LDAP/Oracle Internet Directory-based Identity and Policy Stores for authentication and authorization purposes. These stores in the Shared Environment are used by multiple personal environments set up on developers'

laptops. Like the exploded EAR directories of the deployed applications in the Middleware home, these stores from the shared environment are not intended to be modified by the personal environments that are using them.

Properties and features of the shared environment include:

- The shared environment can be accessed from the personal environment (see Section 2.1.2, "Personal Environment") so developers can reach the MW_HOME.
- The exploded EAR directory in Middleware home can be opened from JDeveloper and a workspace 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.

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

2.1.1.1 Creating the OWSM_MDS Schema

Typically, and particularly in a test environment and a production environment, this schema is in the Oracle Identity Manager (IDM) database. In the case of a development environment, being able to access the Oracle Web Services Manager_Metadata Services (OWSM_MDS) schema provides the same options that Oracle Fusion Applications developers at Oracle have.

There are two options by which the OWSM_MDS schema can be made available to developers:

- The administrator can open the IDM database where the schema already exists.
- The administrator creates the OWSM_MDS schema in the transaction database. See Section 2.1.1.1.1, "How to Create the OWSM_MDS Schema."

2.1.1.1.1 How to Create the OWSM_MDS Schema

This section provides detailed snapshots on how to create the OWSM_MDS schema using the Repository Creation Utility (RCU).

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 *correctly 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 Repository Creation Utility (RCU) 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 a number of RCUs available. To provision Oracle Fusion Applications, you will have created an installer repository. In this repository, you will see the following:

```
installers/
   apps_rcu/
      linux/
         rcuHome_fusionapps_linux.zip
      windows/
         rcuHome_fusionapps_win.zip
   biapps_rcu/
   fmw_rcu/
```

Copy the appropriate .zip file to your system from the `installers/apps_rcu` directory.

Once you get the zip file to your machine, follow these steps:

Linux system

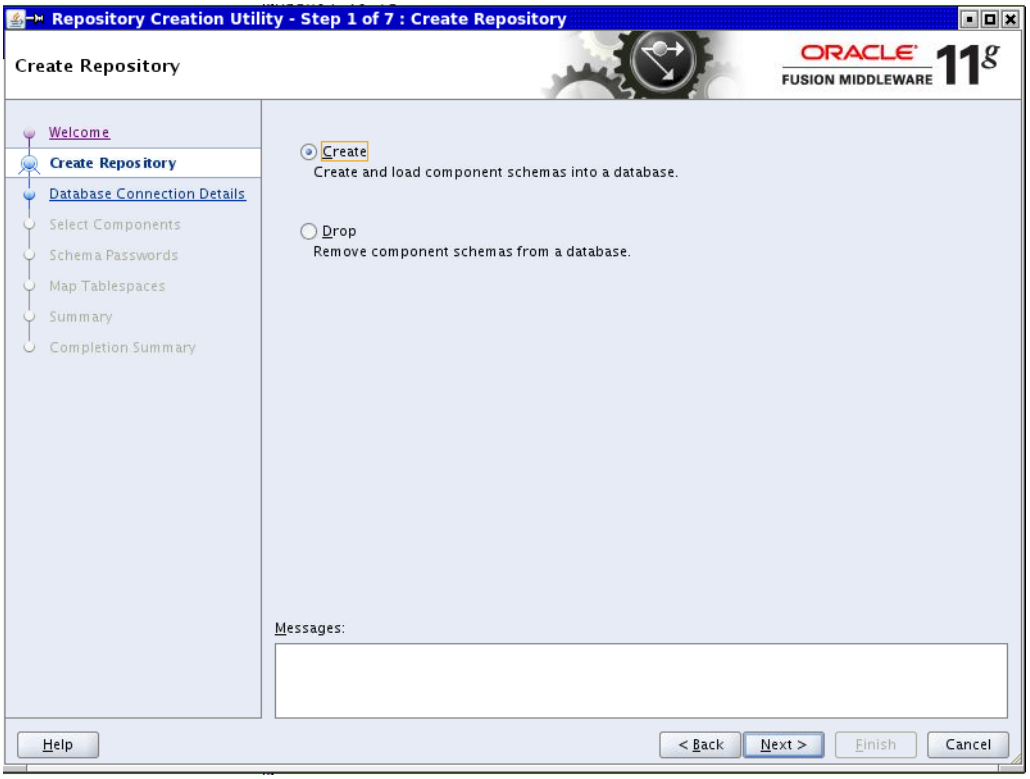
```
% mkdir fa_rcu
% cd fa_rcu
% cp /from_zip_file_location/rcuHome_fusionapps_linux.zip .
% unzip rcuHome_fusionapps_linux.zip
% cd bin
% ./rcu
```

Windows system

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

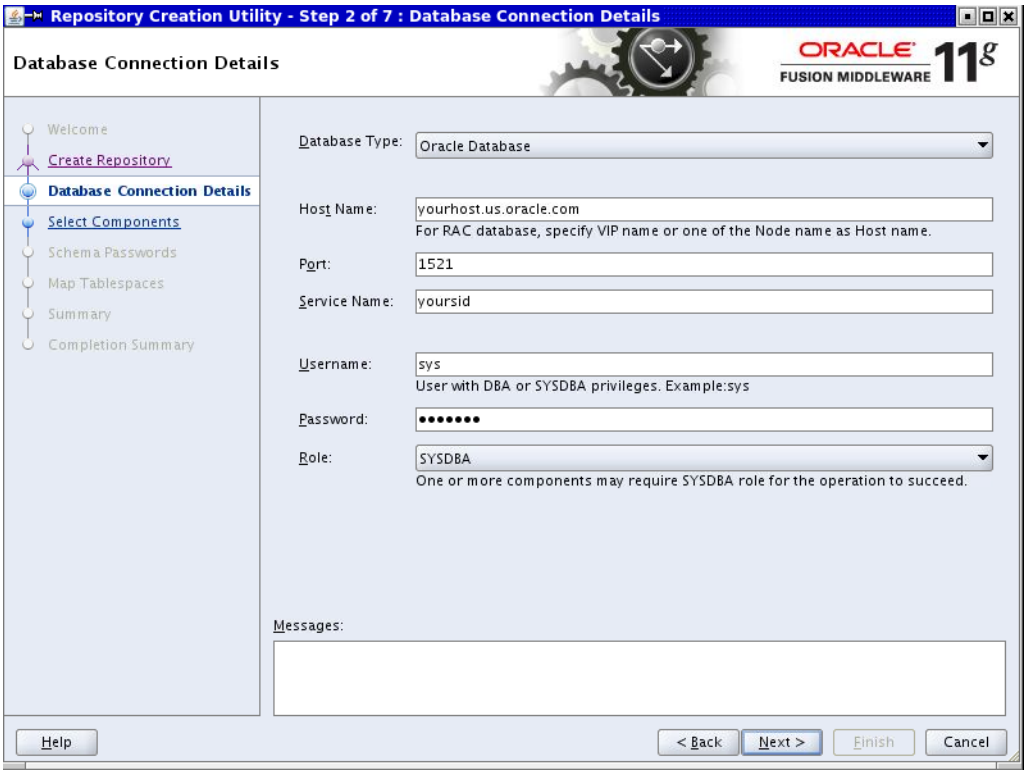
The RCU starts and displays the Create Repository dialog, shown in Figure 2–1.

Figure 2–1 Creating the Repository



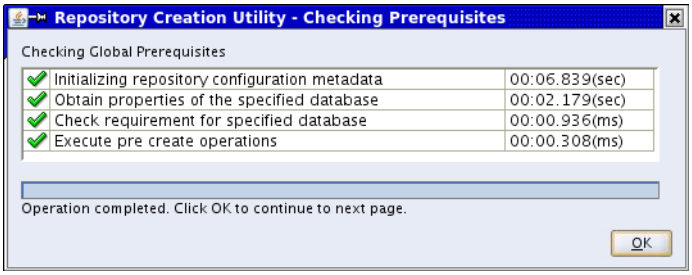
Select **Create** and click **Next** to display the Database Connection Details dialog, shown in Figure 2–2.

Figure 2–2 Creating the Database Connection



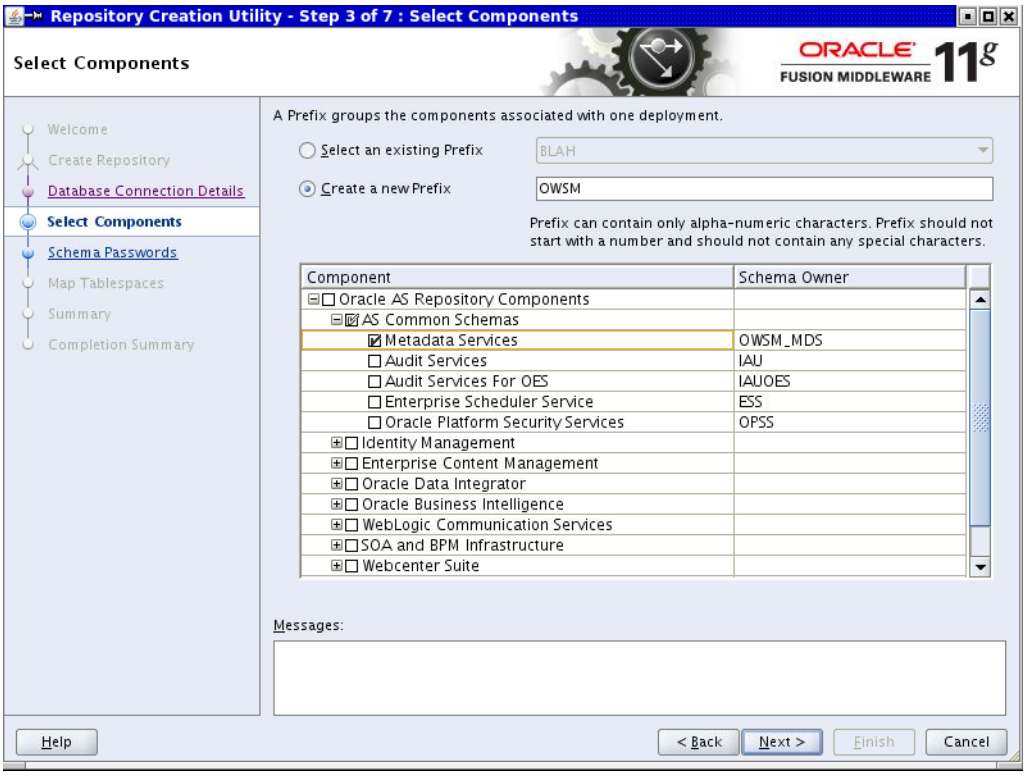
Enter your database connection details and click **Next** to display the Checking Global Prerequisites dialog, shown in Figure 2–3.

Figure 2–3 Checking Global Prerequisites



Click **OK** to display the Select Components dialog, shown in Figure 2–4.

Figure 2–4 Selecting Components

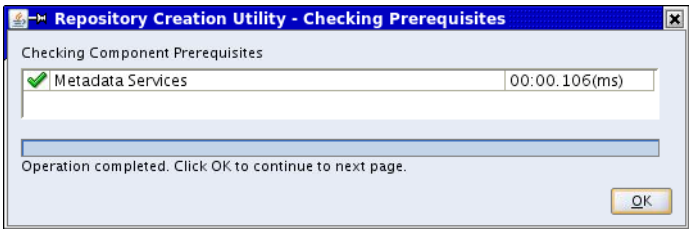


Select **Create a new Prefix** and enter OWSM as the name.

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

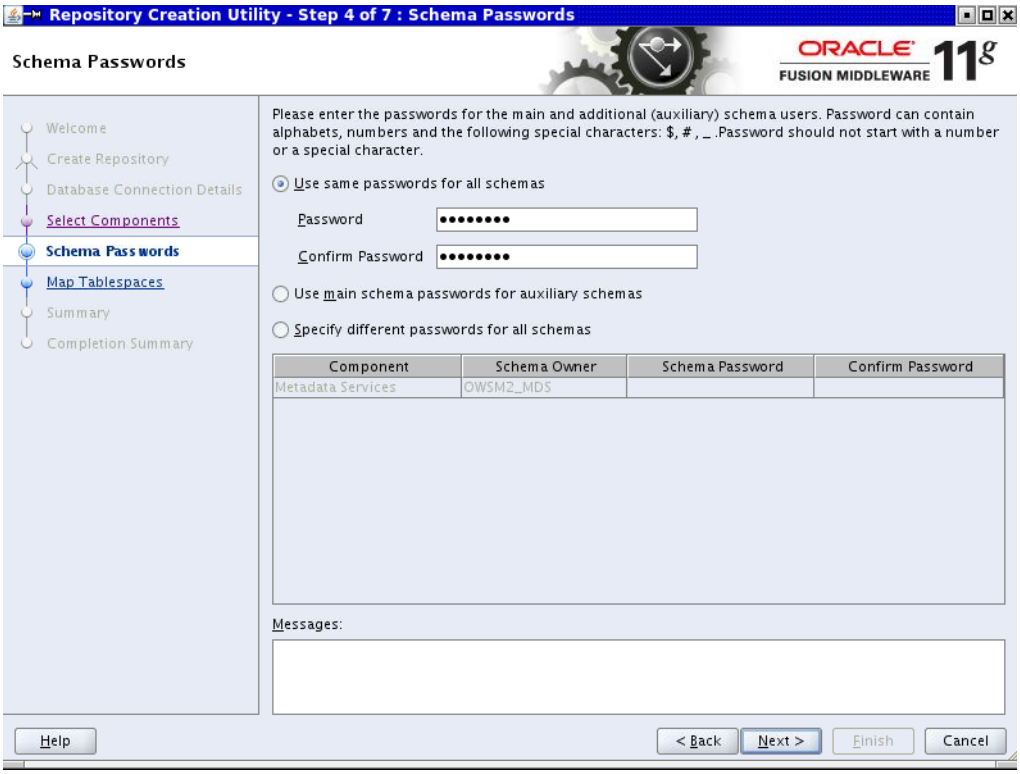
Click **Next** to display the Checking Component Prerequisites dialog, shown in Figure 2–5.

Figure 2–5 Checking Component Prerequisites



Click **OK** to display the Schema Passwords dialog, shown in Figure 2–6.

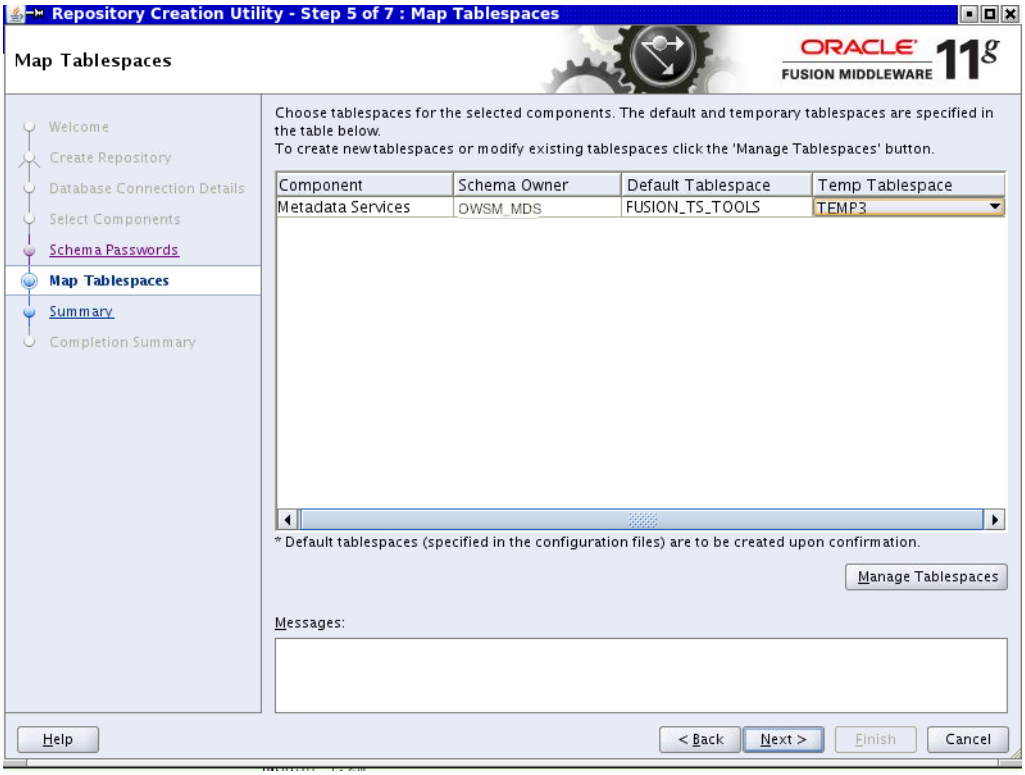
Figure 2–6 Setting Schema Passwords



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 2–7.

Figure 2–7 Mapping Tablespaces

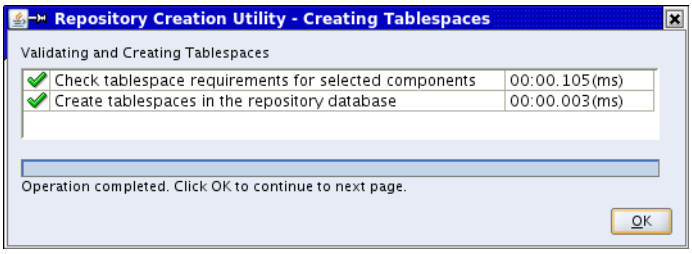


In the Default Tablespace field, select FUSION_TS_TOOLS.

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

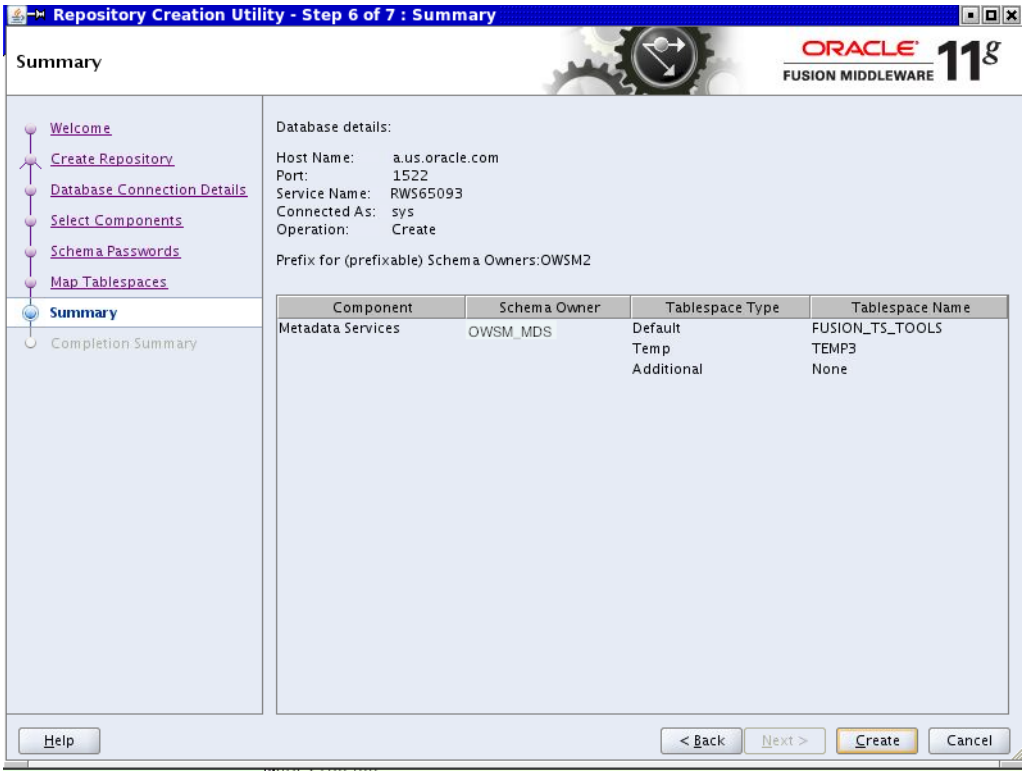
Click **Next** to display the Validating and Creating Tablespaces dialog, shown in Figure 2–8.

Figure 2–8 Validating Created Tablespaces



Click OK to display the Summary dialog, shown in Figure 2–9.

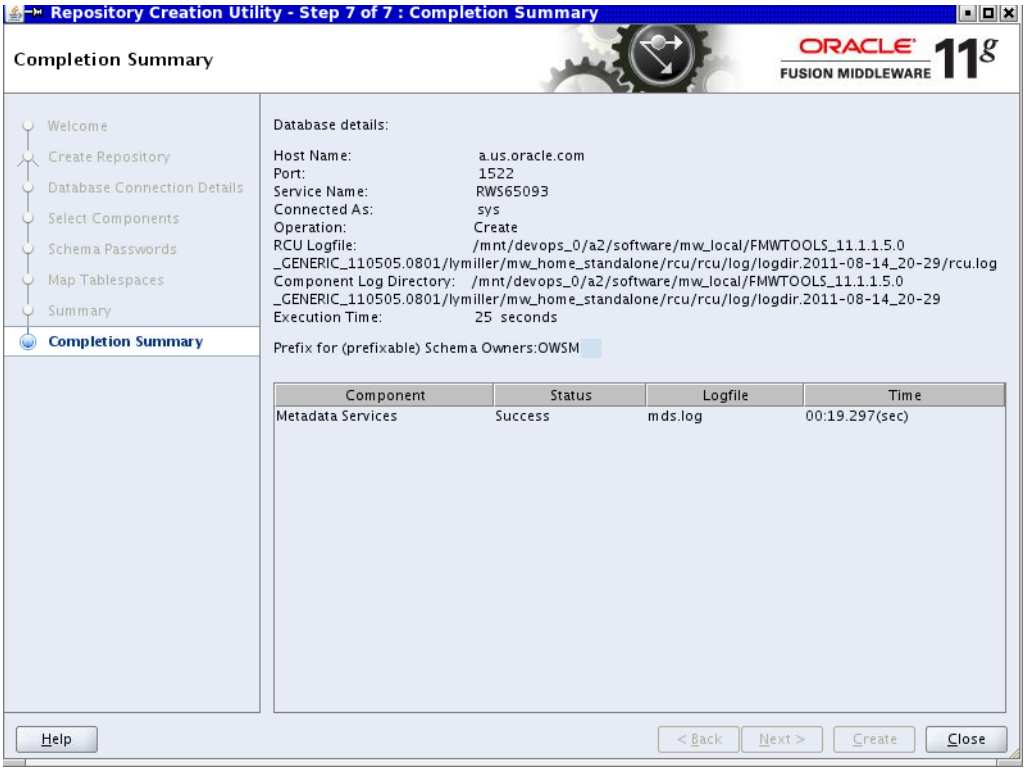
Figure 2–9 Summary



Verify the information and click **Create**.

When the operation has completed successfully, the Completion Summary dialog, shown in Figure 2–10, displays.

Figure 2–10 Completion Summary



Click **Close**.

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

2.1.2 Personal Environment

Each developer has this environment, which 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 without affecting other team members.

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

Once an Integrated or a Standalone Weblogic Server has been launched, you will need to deploy the **oracle.apps.common.resource.ear** shared library. You may need to get the location of the file from an administrator and then use the WebLogic Server Console to deploy it as a shared library using *oracle.apps.common.resource* as the name. See Section 2.2.3.1, "Managing Integrated WebLogic Server."

2.2 Setting Up the JDeveloper-based Personal Environment

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

- Installing JDeveloper
- Installing Extension Bundles
- Applying patches (if necessary)

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 Services (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 Removing 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 running. This process may prevent you from entering a password in the Oracle Fusion Domain wizard or *anywhere a JPasswordField occurs*. You can remove SCIM from your system by executing this command.

```
sudo yum remove scim
```

You also 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 Increasing Open File Limit on Local Linux Servers

This system configuration change is required on local Linux servers to increase the open file limit and resolve a number of JDeveloper, WebLogic Server and other "Too many files" errors when doing a build or merge.

- Add these two instructions to the **/etc/security/limits.conf** file:
  - soft nofile 8192
  - hard nofile 8192
- Restart the machine to have these values take effect.
- 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

JDeveloper is supplied on the **Oracle JDeveloper 11g and Oracle Application Development Framework 11g (11.1.1.5.3)** disk. JDeveloper support files, such as extensions, are supplied on the **Oracle Fusion Applications Companion 11g (11.1.1.5.3)** disk. Your administrator may choose to make the contents of the disks available on a shared directory that will have the same directory structure as the disks. We strongly recommend that the administrator make the contents of the Oracle Fusion Applications Companion 11g (11.1.1.5.3) disk available on a shared directory. The directions in this section assume that you are installing from the disk.

Install the Studio edition of JDeveloper from the top-level directory of the Oracle JDeveloper 11g and Oracle Application Development Framework 11g (11.1.1.5.3) disk.

For Windows, you can use the **jdevstudio11115install.exe** installer. For Linux, you can use the **jdevstudio11115install.bin** installer. If you decide to use the generic **jdevstudio11115install.jar** installer, you *must* first install JDK 6 Update 24 from the Oracle Technical Network and then install JDeveloper using the generic installer.

The installation will let you specify the directory into which to install JDeveloper. This installation directory is MW_HOME.

When you have started JDeveloper, you will need to install the extension bundles from the `fusion_apps_extensions` directory on the Oracle Fusion Applications Companion 11g (11.1.1.5.3) disk. See Section 2.2.1.5, "Setting Up the JDeveloper-based Development Environment."

For more installation information, see the *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper (Oracle Fusion Applications Edition)*.

2.2.1.4 Adding 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/directory
2
```

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. You will have to get one or more paths from an administrator. The administrator can use these steps to locate the JAR files:

- 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 to find it.

2.2.1.5 Setting 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 2.4.3 or later on your machine. If you do not, install Python version 2.4.3 or newer.
2. Set these environment variables:

   csh commands:

   ```
   setenv PATH /path/to/python/bin:$PATH
   setenv MW_HOME /path/to/JDeveloper/installation/directory
   setenv JAVA_HOME $MW_HOME/jdk160_24
   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:$PATH
   export MW_HOME=/path/to/JDeveloper/installation/directory
   export JAVA_HOME=$MW_HOME/jdk160_24
   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 PATH=\path\to\python\bin:%PATH%
   set MW_HOME=\path\to\JDeveloper\installation\directory
   set JAVA_HOME=%MW_HOME%\jdk160_24 (Important: For Windows 64-bit, JAVA_HOME
   should point to a 64-bit JVM.)
   set PATH =%JDK_HOME%\bin:%PATH%
   set JDEV_USER_HOME=\path\to\a\directory
   (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
   ```

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

Save and close **jdev.conf**.

Table 2–1 VMOptions in jdev.conf

Name	Value	Comments
`-Dide.extension.extra.search.path`	`/x/y/z`	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.
`-DURLChooser.forceUseList`	true	Optional.
`-DURLChooser.disableCompletionPopup`	true	Optional.
`-DUNIX_WEB_BROWSER`	/usr/bin/firefox	Location of the browser executable specific to the environment at the customer's site.
`-Djbo.SecondaryADFLibVisible`	true	
`-Ddeployment.jario.writepolicy`	recreate	
`-Doracle.webcenter.portlet.enableApplicationStriping`	true	
`-Doracle.webcenter.portlet.dt.excludeExportSet`	true	
`-Dadflib.project.open.refresh`	false	
`-Djdev.wlst.env.vars`	HOME,JDEV_USER_HOME	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.
`-XX:MaxPermSize`	512M	
`-Xmx1024M`		
`-Xms256M`		
`-XX:+DisableExplicitGC`		

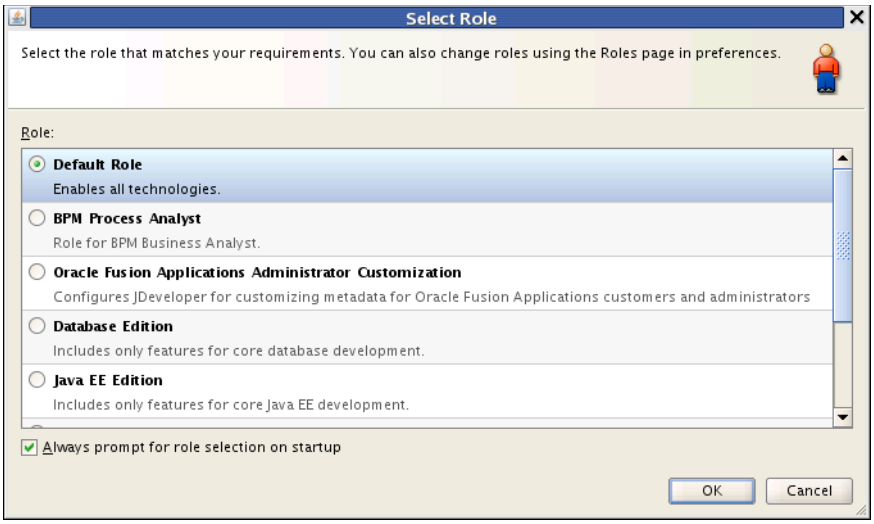
5. Start JDeveloper.

```
jdev &
```

If you require more details about the JDeveloper startup, you can set the VERBOSE environment variable. If you are using csh, the command is `setenv VERBOSE TRUE`. If you are using bash, the command is `export VERBOSE=TRUE`.

When prompted, select the **Default Role**, as shown in Figure 2–11.

Figure 2–11 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 2–2 JDeveloper Roles

Role	Description
Default Role	This role allows you to access all of JDeveloper's features. The other roles provide subsets of these features.
Oracle Fusion Applications Administrator Customization	This is the main customization role for Oracle Fusion Applications customers. **Important:** You *must* use this Role for customizing SOA Composites.
Oracle Fusion Applications Developer	This is for Oracle Fusion Applications developers to use to build new applications.
Database Edition	This gives you access to just the core database development tools.
Java EE Edition	This includes only features for core Java EE development.
Java Edition	This includes only features for core Java development.

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

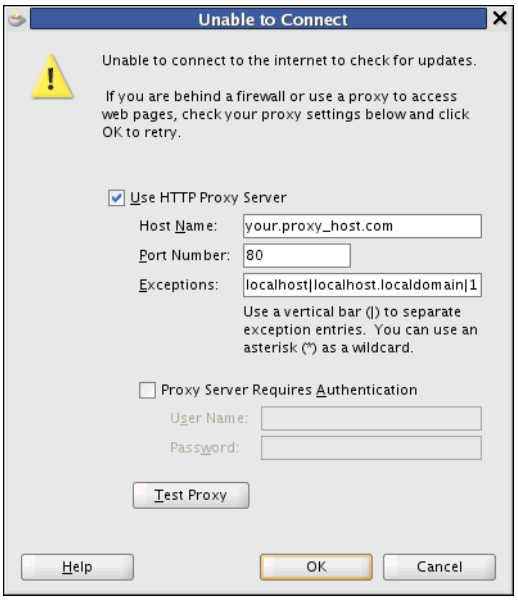
6. Install the **Fusion Apps Development Environment** extension bundle, an all-encompassing JDeveloper bundle that is specific for Oracle Fusion Applications. To install the bundle:

   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 2–12.

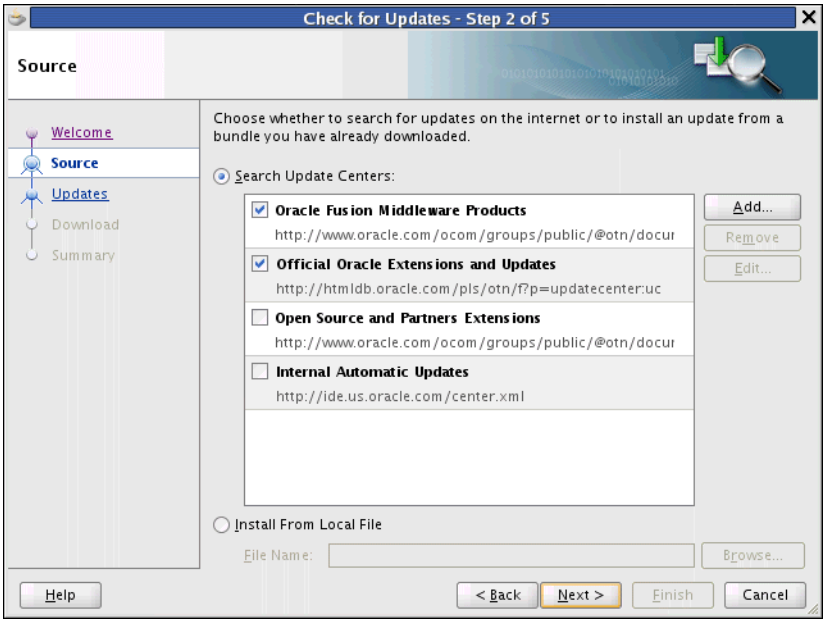
Figure 2–12 Completing the Proxy Setup Dialog



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

d. Click **OK**. The Source dialog, shown in Figure 2–13, displays.

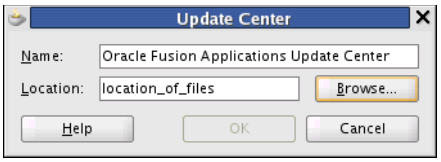
Figure 2–13 Available Update Centers



e. Unselect any marked Update Centers.

f. Click **Add** to display the Update Center dialog, shown in Figure 2–14.

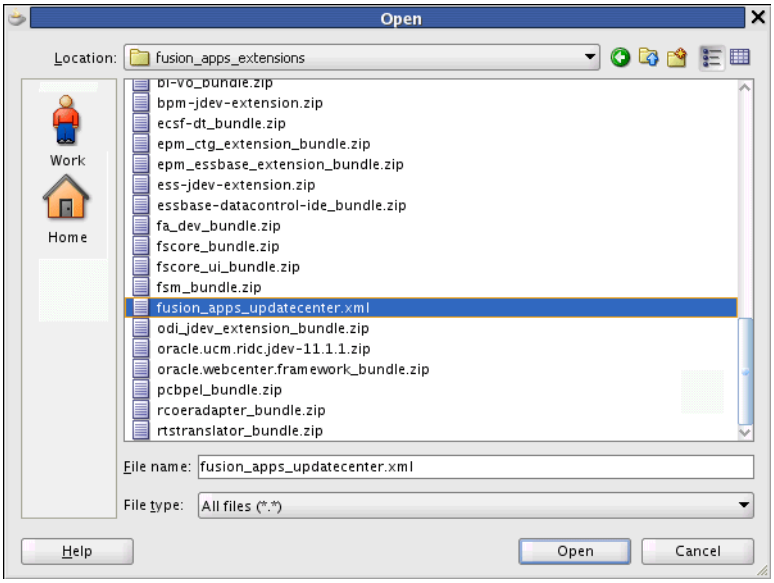
Figure 2–14 Adding a New Update Center



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 2–15. The location of this file in on the **Oracle Fusion Applications Companion 11g (11.1.1.5.3)** disk in the `fusion_apps_extensions` directory, or on a shared directory provided by the administrator.

Figure 2–15 Browsing for the Update File

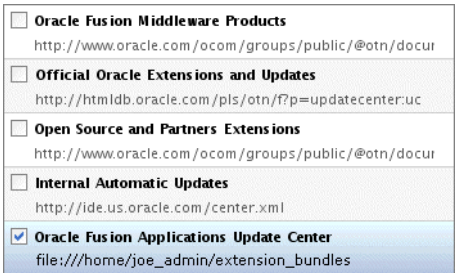


i. Click **Open**.

j. Click **OK**.

k. Select the newly-defined Update Center, as shown in Figure 2–16.

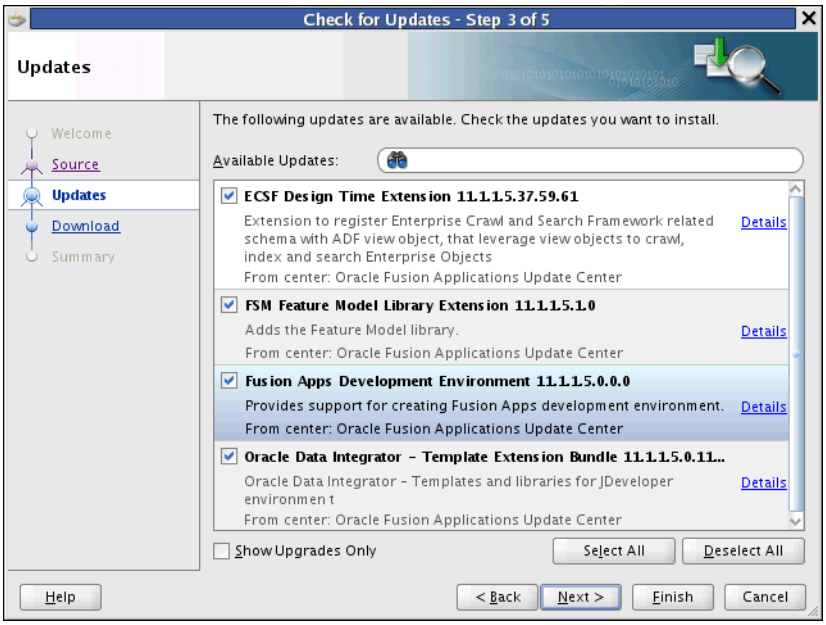
Figure 2–16 Selecting the New Update Center



Note that *only* the new Update Center should be selected.

l. Click **Next** to display the Updates dialog, shown in Figure 2–17. Note that, initially, no updates will be selected.

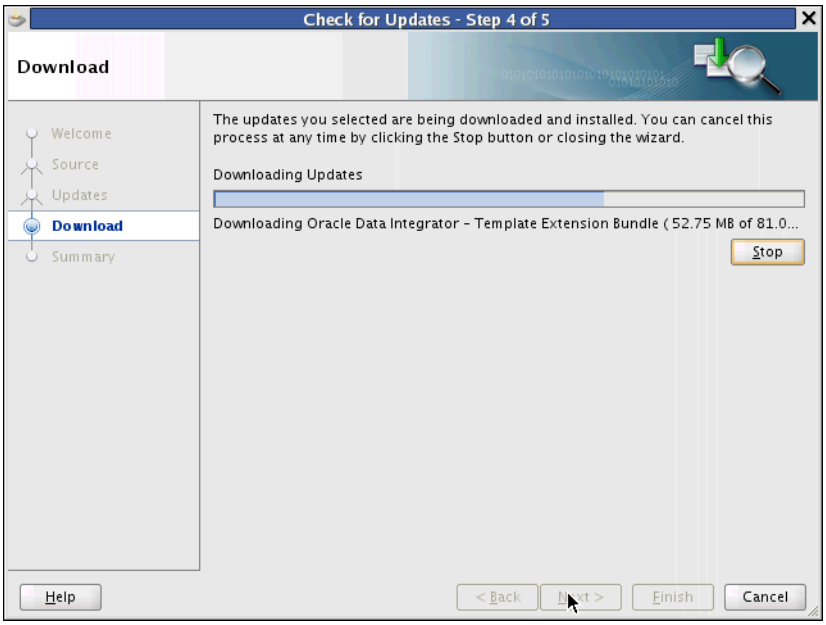
Figure 2–17 Selecting the Updates



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

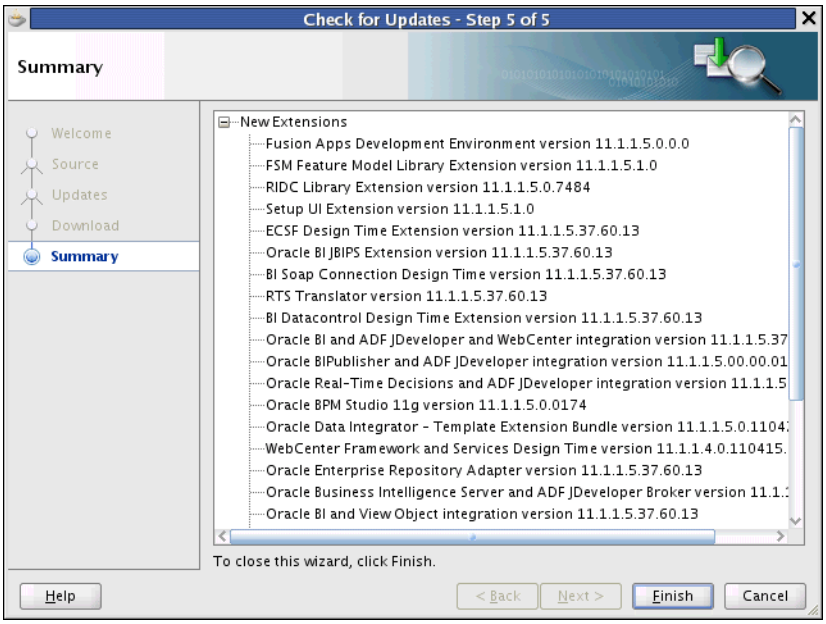
n. Click **Next** to display the Download dialog and start the download, as shown in Figure 2–18.

Figure 2–18 Downloading the Updates



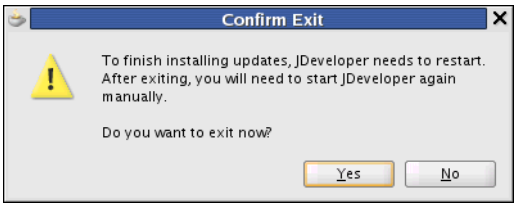
When the download finishes, the Summary dialog, similar to that shown in Figure 2–19, displays automatically.

Figure 2–19 Summary Dialog



o. Click **Finish**. If you are using JDeveloper on a Linux system, the Confirm Exit prompt shown in Figure 2–20 displays.

Figure 2–20 Confirm Exit Prompt

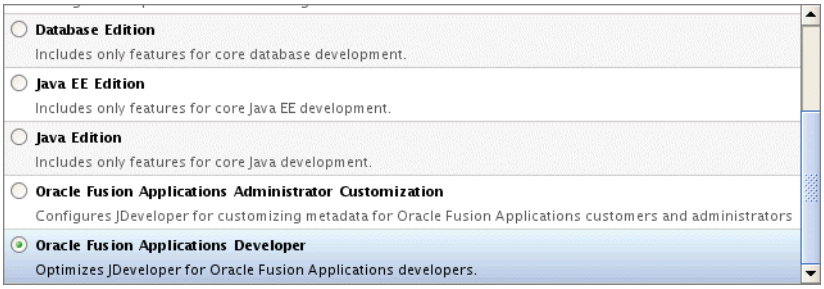


p. Click **Yes** to exit JDeveloper.

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 2–21. The restart results in installing the extension bundles that were downloaded.

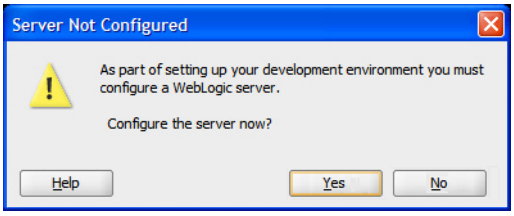
Figure 2–21 Selecting the Oracle Fusion Applications Developer Role



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 2–22. Click **Yes**. See Section 2.2.2, "How to Use the Oracle Fusion Domain Wizard."

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 Section 2.2.1.7, "Distributing the fusion_apps_wls.properties and cwallet.sso Files."

Figure 2–22 WebLogic Server Not Configured Prompt



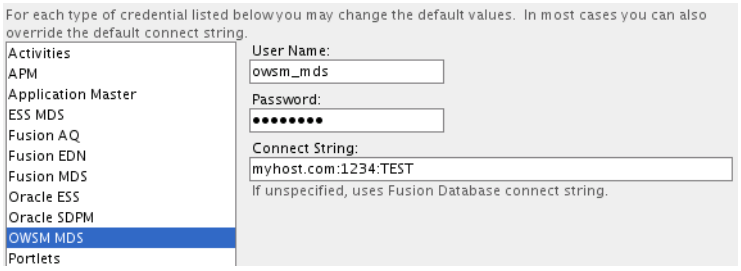
Once JDeveloper is up, the environment should have all the components in the correct locations. You should be able to create new, or customize existing, Oracle Fusion applications.

2.2.1.6 Using the OWSM_MDS Schema

There are two options for how a developer can use the OWSM_MDS 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 2–23 shows how the wizard's Database dialog will be completed.

Figure 2–23 Setting Up the OWSM_MDS Schema



- The administrator will use the Repository Creation Utility (RCU) to create the MDS schema, described in Section 2.1.1.1.1, "How to Create 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.

2.2.1.7 Distributing 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 **$JDEV_USER_HOME/system11.1.1.5.xx.yy.zz** folder. The administrator can distribute the entire **o.jdevimpl.rescat2** sub-folder to the development team. The developers can install JDeveloper and install the bundles. The developers then can copy the entire **o.jdevimpl.rescat2** sub-folder under their own **$JDEV_USER_HOME/system.11.1.1.5.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. Note that the properties file and the **cwallet.sso** file *must* be in the same location.

The wizard can be run multiple times to change properties in the file. If certain critical properties are changed, the domain may 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:

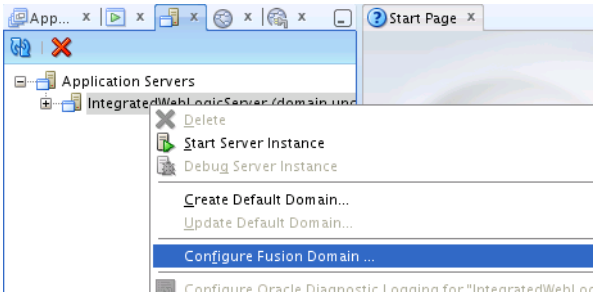
- The **fusion_apps_wls.properties** file is not found when JDeveloper starts. The name of the file defaults to **fusion_apps_wls.properties**, and the location defaults to the `system11.1.1.xx.yy.zz/o.jdevimpl.rescat2` directory.
- The **fusion_apps_wls.properties** file is not found when you select to **Run** an application from JDeveloper, or you select to **Start Server Instance** from JDeveloper.

To start the wizard manually from within JDeveloper:

- Select **View > Application Server Navigator**.
- Expand the **Application Servers** node.

- Right-click **IntegratedWebLogicServer** and select the **Configure Fusion Domain** option, as shown in Figure 2–24.

Figure 2–24 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:

```
[domain]
domainType=adminall
```

Table 2–3 Properties to be Captured in the Oracle Fusion Domain Wizard

Property Name	Standalone/ Integrated	Required	Default	Comments
`domainType`	Standalone	Yes	standalone	Valid values are: **adfess**: Admin server only for ADF and ESS **adminsoa**: Admin server (EM) and SOA managed server **adminall**: Combination of standalone and adminsoa (ADF,ESS,SOA) **adminessadf**: Admin server for ADF and ESS managed server **standalone**: Default setting. This is an admin server only; not a managed server.
`domainName`	Standalone	Yes	fusion_domain	
`domainDir`	Standalone	Yes		The location in the file system in which the domain will be created. 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.

Table 2–3 (Cont.) Properties to be Captured in the Oracle Fusion Domain Wizard

Property Name	Standalone/ Integrated	Required	Default	Comments
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. 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	
wlPassword	Both	Yes	weblogic1	
ldapHost	Both	Yes		This is a string value similar to: ldaphostname.yourcompany.com.
ldapPort	Both	Yes		Example: 3060
ldapUser	Both	Yes		Example: cn=wlsproxyuser
ldapPass	Both	Yes		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.
ldapUserDN	Both	Yes		Example: cn=users,dc=us,dc=yourcompany,dc=com
ldapGroupDN	Both	Yes		Example: cn=groups,dc=us,dc=yourcompany,dc=com
ldapSSLEnabled	Both	No	false	true or false
opssHost	Standalone	No		Optional separate OPSS store. This is a string value similar to: opsshostname.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**).
opssPort	Standalone	No		Example: 3061
opssUser	Standalone	No		Example: cn=wlsproxyuser

Table 2–3 (Cont.) Properties to be Captured in the Oracle Fusion Domain Wizard

Property Name	Standalone/ Integrated	Required	Default	Comments
opssPass	Standalone	No		Example: welcome2 **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.
familyName	Both	Yes		Family names are: COMMON, IC, HCM, FIN, PRC, PRJ, SCM, and CRM.
fusionDb - connect string	Both	Yes		JDBC connect string. Note: This is split into sub-fields: ■ **fusionDbUser** ■ **fusionDbPassword** ■ **fusionDbHost** ■ **fusionDbPort** ■ **fusionDbSid** This is applicable to these schemas: ■ **activityDb** ■ **apmDb** ■ **AppMasterDb** ■ **essMdsDb** ■ **fusionAq** ■ **fusionEdn** ■ **fusionMds** ■ **oraessDb** ■ **orasdpmDb** ■ **owsmMdsDb** ■ **portletDb** ■ **soadatasrcDb** ■ **soaMdsDb** ■ **wcDb**

Table 2–3 (Cont.) Properties to be Captured in the Oracle Fusion Domain Wizard

Property Name	Standalone/ Integrated	Required	Default	Comments
`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.
`fusionAqUser` - connect string	Both	Yes, but the database connection is the same as for fusionDb.		You can connect to the fusionDB database as `fusion_aq`. Your system administrator will provide the connection details and schema password to be used.
`fusionEdnUser` - connect string	Both	Yes, but the database connection is the same as for fusionDb.		**Note**: CRM will be replaced with the familyName that is passed in. 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.
`fusionMdsUser` - connect string	Both	Yes, but the database connection is the same as for fusionDb.		You can connect to the fusionDB database as `fusion_mds`. Your system administrator will provide the connection details and schema password to be used.
`oraEssDbUser` - connect string	Both	No		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.
`orasdpmDbUser` - connect string	Both	No		You can connect to the fusionDB database as `fusion_orasdpm`. Your system administrator will provide the connection details and schema password to be used.
`owsmMdsDbUser` - connect string	Both	No		You can connect to the fusionDB database as `owsm_mds`. Your system administrator will provide the connection details and schema password to be used.

Table 2–3 (Cont.) Properties to be Captured in the Oracle Fusion Domain Wizard

Property Name	Standalone/ Integrated	Required	Default	Comments
`portletDbUser` - connect string	Both	No		You can connect to the fusionDB database as `fusion_portlet`. Your system administrator will provide the connection details and schema password to be used.
`soadatasrcDbUser` - connect string	Both	No		Family name used in prefix. 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.
`soaMdsDbUser` - connect string	Both	No		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.
`wcDbUser` - connect string	Both	No		You can connect to the fusionDB database as `fusion_webcenter`. Your system administrator will provide the connection details and schema password to be used.

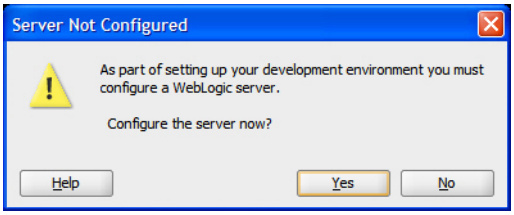
2.2.2.1 Creating the Properties File for Default Integrated Server

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 Section 2.2.1.7, "Distributing the fusion_apps_wls.properties and cwallet.sso Files."

The wizard will start automatically if the **fusion_apps_wls.properties** file is not found when you start JDeveloper. You also can start the wizard manually. See Section 2.2.2, "How to Use the Oracle Fusion Domain Wizard."

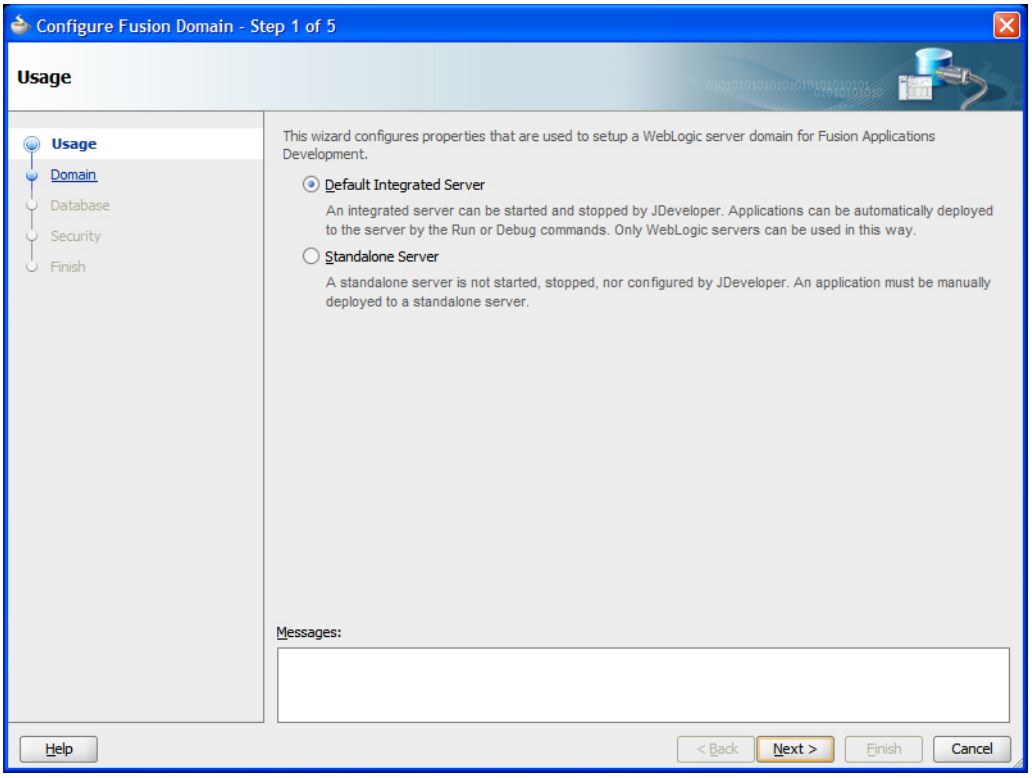
If the **fusion_apps_wls.properties** file is not found when you start JDeveloper, the prompt shown in Figure 2–25 displays.

Figure 2–25 JDeveloper Startup Server Configuration Prompt



Click **Yes** to launch the wizard and display the Usage page, as shown in Figure 2–26.

Figure 2–26 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 Section 2.2.2.2, "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 using the **fusion_apps_wls.properties** file as input. See Section 2.3, "Setting Up the Personal Environment for Standalone WebLogic Server."

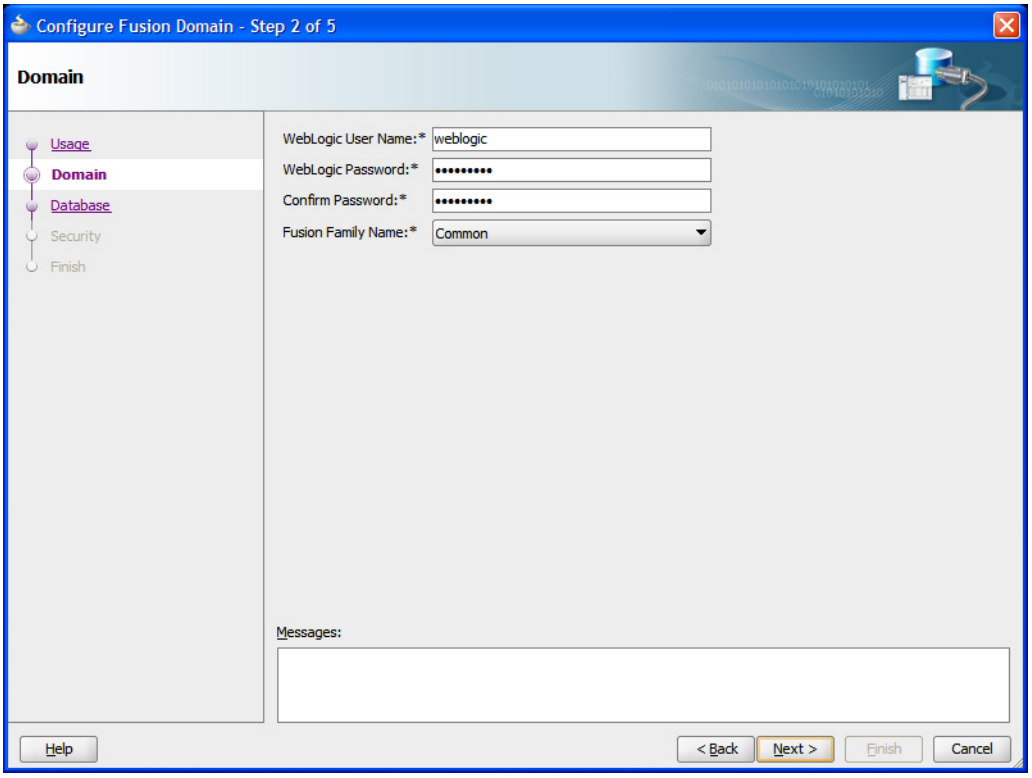
- **Messages**

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

Further wizard pages depend on the selected Usage. The flow for the **Default Integrated Server** selection is covered first.

When you select the **Default Integrated Server** Usage option and click **Next**, the Domain dialog, shown in Figure 2–27, displays.

Figure 2–27 Configuring the 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.

- **WebLogic User Name**

  This value defaults to **weblogic**. Change it if necessary.

- **WebLogic Password / Confirm Password**

  The password requires at least one numeral. It defaults to **weblogic1**.You can change it if necessary. 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**

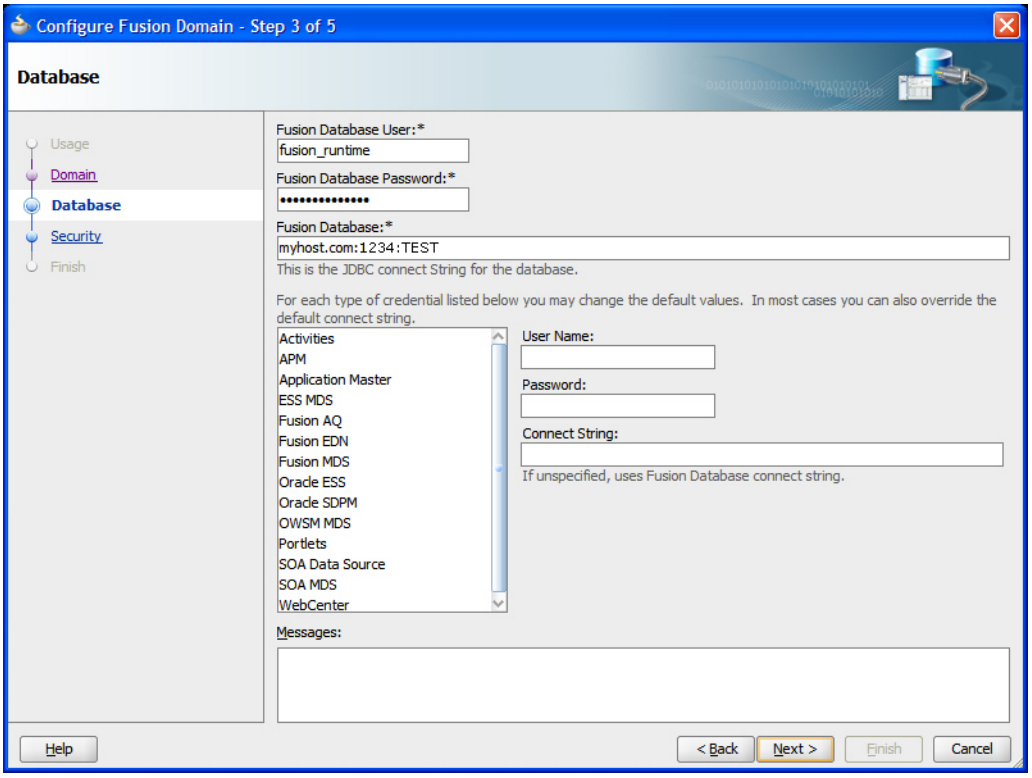
  The default is Common to signify the common domain. If necessary, from the drop down list, select the Fusion Family Name, such as Customer Relationship Management or Financials.

- **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 2–28

Figure 2–28 Configuring the Database



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

Notes:

- 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. For example, if the password of the `Activities` credential happens to be `fooBar812` instead of `fusion_activities` (which is the schema name), you should select the `Activities` credential from the list and provide the password for the `fusion_activities` schema as `fooBar812` in the Password field. The Username field will contain `fusion_activities`. You do not have to provide anything in the Connect String field for this schema, as it will be in the same database.
- 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. Once 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**

  A number of 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 Section 2.1.1.1, "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**

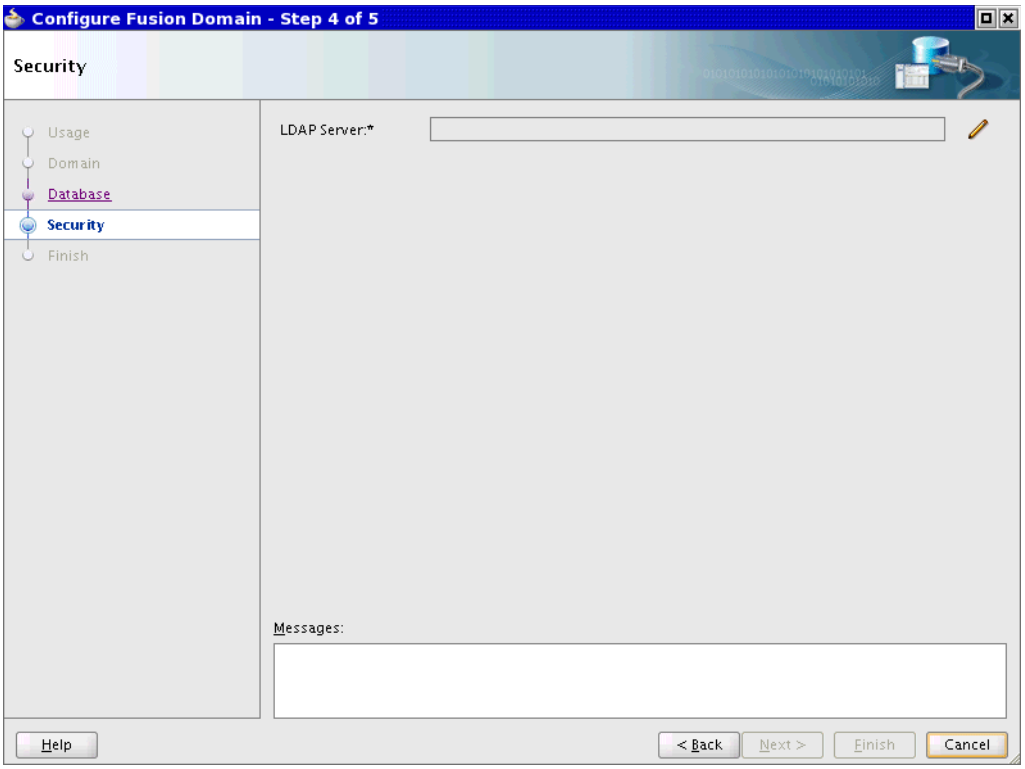
    This field corresponds to the Fusion Database field.

- **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 2–29.

Figure 2–29 Configuring Security for Integrated Domain Server

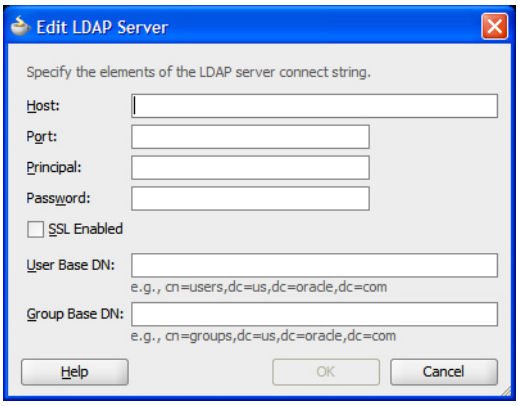


If the **fusion_apps_wls.properties** file already exists and is in place, the fields will show the values that are in the file.

- **LDAP Server**

  This field cannot be edited directly. Click the edit icon to display the Edit LDAP Server dialog shown in Figure 2–30.

Figure 2–30 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 different name and location from all other objects in Active Directory.

    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**

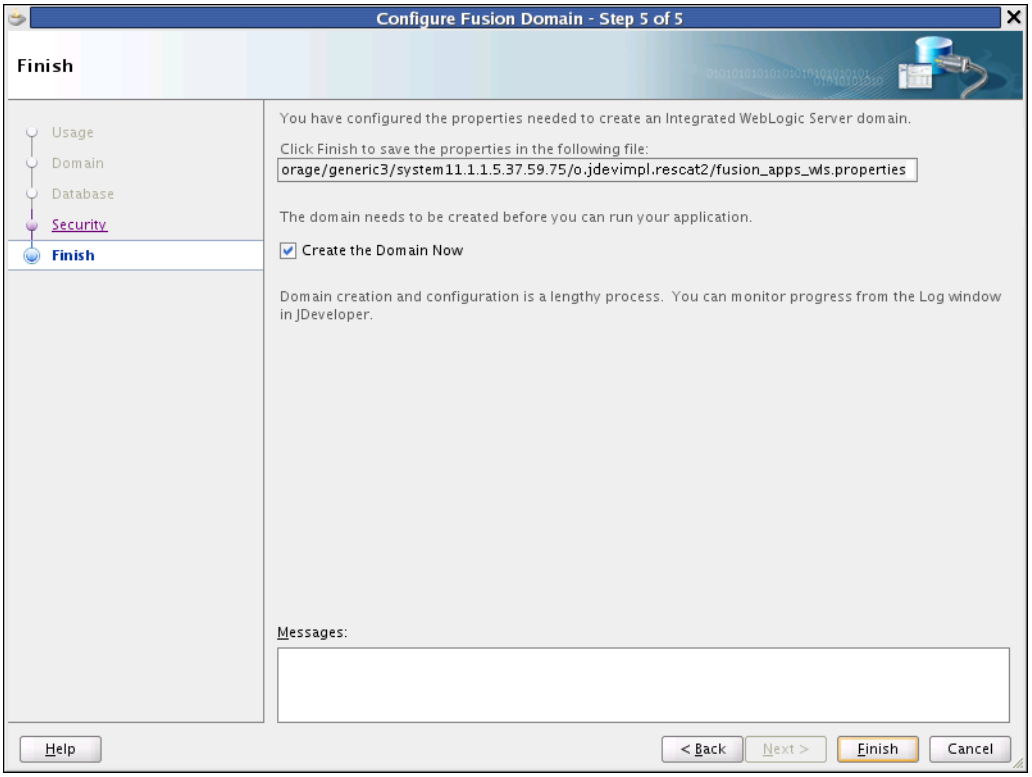
    Enter the Group DN based your LDAP. A sample Group DN resembles `cn=groups,dc=us,dc=your_company,dc=com`.

- **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 2–31.

Figure 2–31 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**

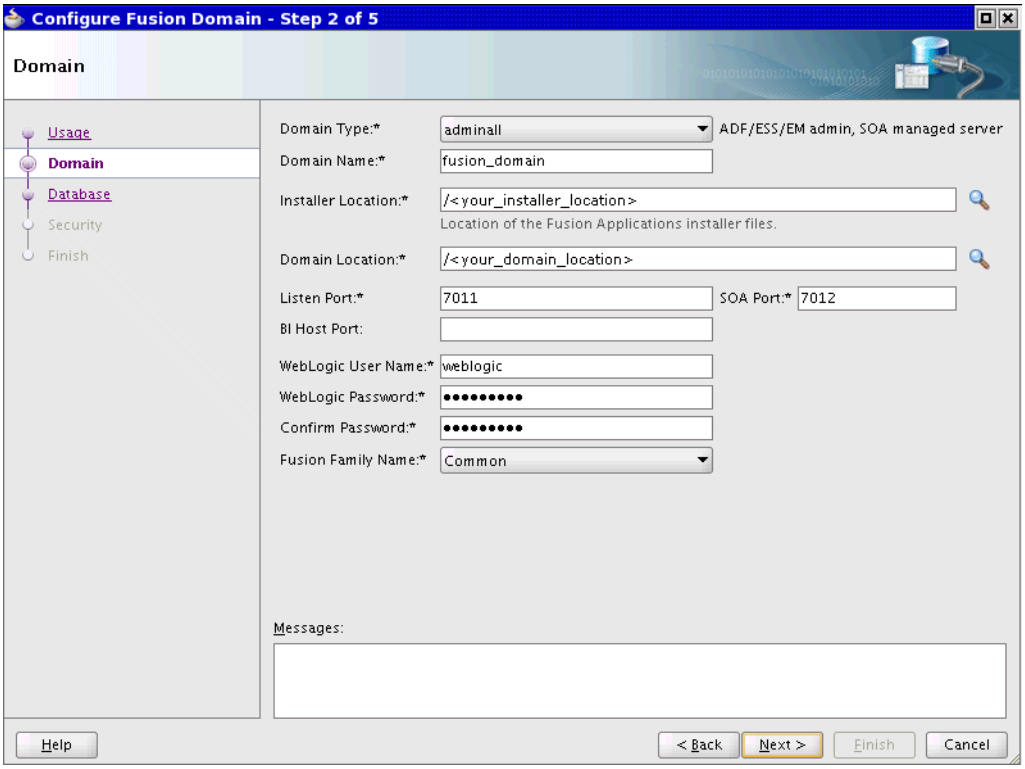
  This defaults to Yes (checked). When selected and you click **Finish**, the Integrated WebLogic Server domain will be created so you can test your applications by selecting one of the JDeveloper **Run** options.

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 Completing 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 2–32, displays.

Figure 2–32 Configuring 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 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 ADF/Oracle Enterprise Scheduler and managed server soa_server1 for Oracle SOA Suite.
  - **adminessadf**: Admin server for 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.

  - 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.

- **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. It defaults to **weblogic1**.You can change it if necessary. 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**

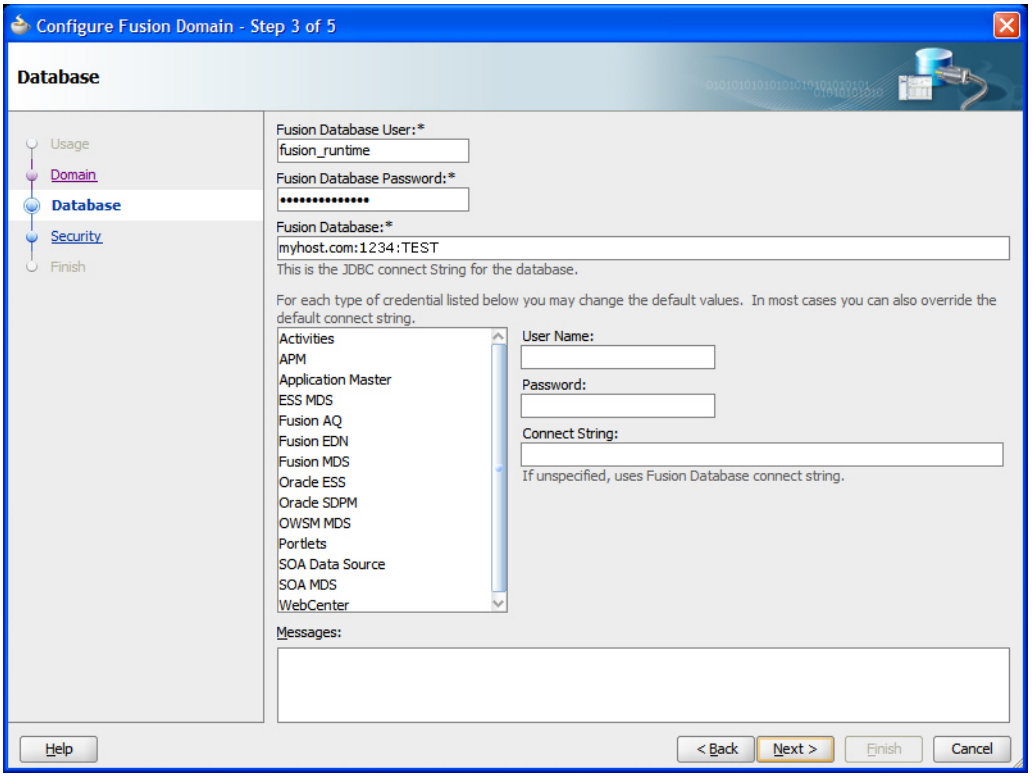
The default is Common to signify the common domain. If necessary, from the drop down list, select the Fusion Family Name, such as Customer Relationship Management or Financials.

- **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 2–33

Figure 2–33 Configuring the Database



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

Notes:

- 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. For example, if the password of the `Activities` credential happens to be `fooBar812` instead of `fusion_activities` (which is the schema name), you should select the `Activities` credential from the list and provide the password for the `fusion_activities` schema as `fooBar812` in the Password field. The Username field will contain `fusion_activities`. You do not have to provide anything in the Connect String field for this schema, as it will be in the same database.
- 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. Once 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. 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**

  A number of 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 Section 2.1.1.1, "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**

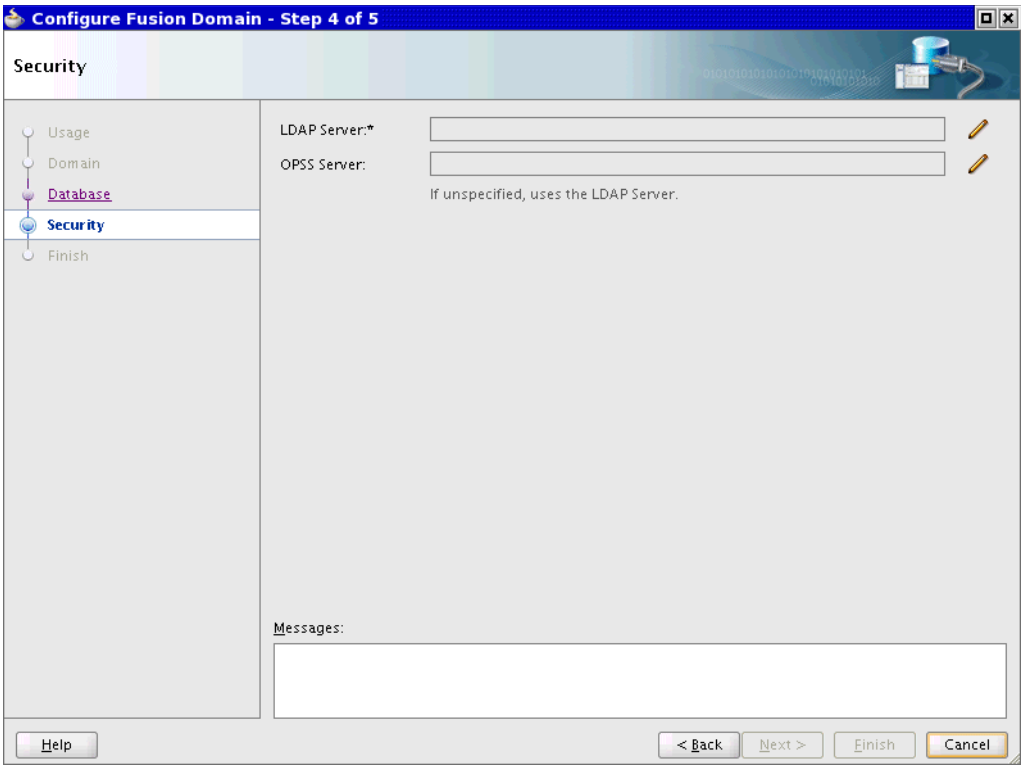
    This field corresponds to the Fusion Database field.

- **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 2–34.

Figure 2–34 Configuring Security for Standalone Server

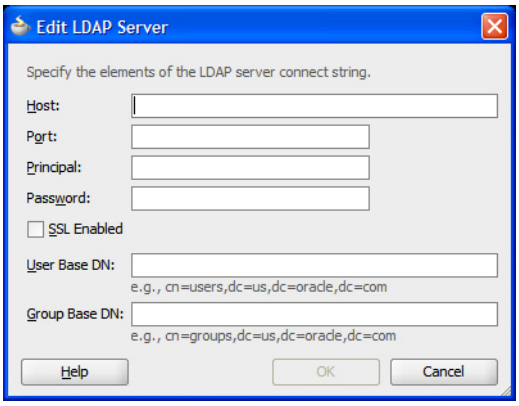


If the **fusion_apps_wls.properties** file already exists and is in place, the fields will show the values that are in the file.

- **LDAP Server**

  This field cannot be edited directly. Click the edit icon to display the Edit LDAP Server dialog shown in Figure 2–35.

Figure 2–35 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 different name and location from all other objects in Active Directory.

  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.

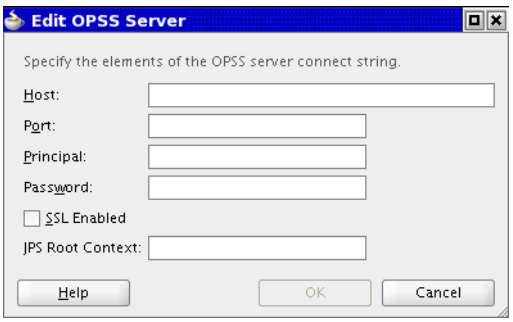
- **Group Base DN**

  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**.

Figure 2–36 Editing the OPSS 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=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**

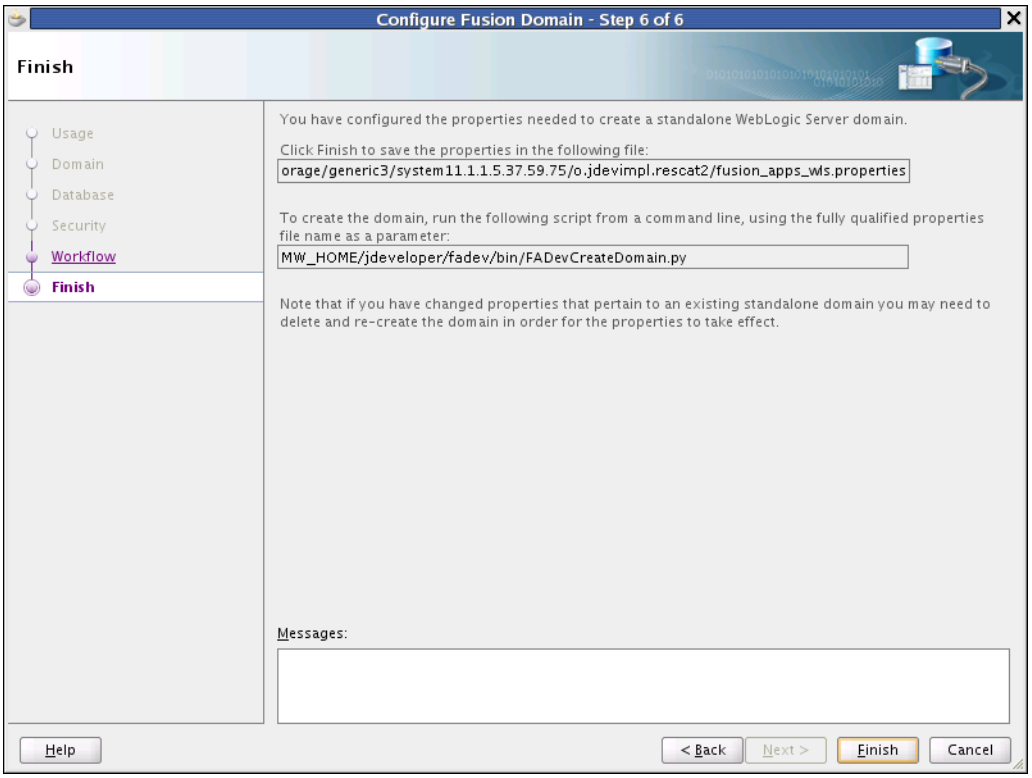
  This value defaults to not enabled. Select this check box if you want to enable SSL.

- **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 2–37.

Figure 2–37 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 Section 2.3.1, "How to Create 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

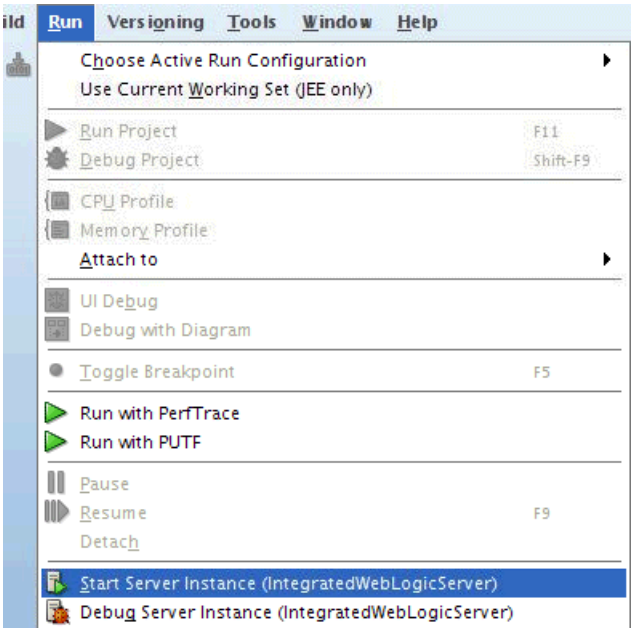
Integrated WebLogic Server and the deployed applications are separate entities. You can start Integrated WebLogic Server before running or deploying any applications.

Starting 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 2–38.

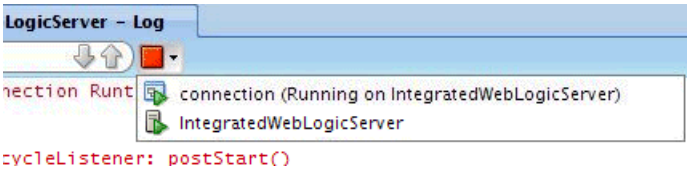
Figure 2–38 Start Server Using Start Server Instance



Stopping 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 2–39.

Figure 2–39 Stopping Integrated WebLogic Server or the Application

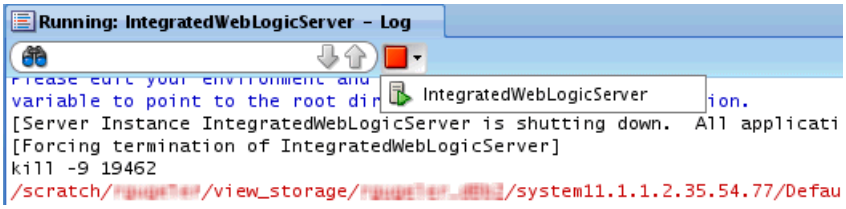


If you select the **connection** option, the application will be undeployed and the server will remain running.

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 2–40.

Figure 2–40 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.

Manually killing the process

1. From a terminal window, execute this command.

   ```
   /usr/sbin/lsof -i -P | grep 7101
   ```

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

2. Kill the process. For instance, if the process is 15846, you would execute this command:

   ```
   kill -9 15846
   ```

2.2.3.1 Managing Integrated WebLogic Server

The WebLogic Server Console can be deployed and 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`.

The default username and password for the Integrated WebLogic Server Console application are *weblogic / weblogic1*.

2.3 Setting Up the Personal Environment for Standalone WebLogic Server

While the JDeveloper-based environment with Integrated Weblogic Server is useful in creating and validating customizations to the ADF artifacts, it cannot be used to validate SOA customizations. Also, anything that relies on SOA, such as BPM and ESS, will need the standalone environment.

To create this environment, you need 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 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. As a result, 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 once they are successfully tested by developers.

See "Managing the Metadata Repository" in the *Oracle Fusion Middleware Administrator's Guide*.

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 has 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.

2.3.1 How to 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.

Notes:

- The installer repository on Windows *must* be a local or a mapped drive.
- Windows and Linux operating systems must be 64-bit.

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

2.3.1.1 Creating 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.

Important: You *must* use the Oracle Fusion Applications Administrator Customization role for customizing SOA Composites.

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 *Oracle Fusion Middleware Administrator's Guide*.

2.3.1.2 Setting 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. You will need installer files from the provisioned environment. For that, you may have to talk to the site administrator to make the installer files available.
2. Update the properties file that was created for Integrated WebLogic Server so that it can also 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 Section 2.2.2, "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.

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

3. 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
     ```

     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
     ```

     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
     ```

   - mkdir /path_to/FAStandaloneWork
   - chmod +x $JDEV_MW_HOME/jdeveloper/fadev/bin/*.py
   - Create a lightweight MW_HOME by running the **FADevInstallMwHome.py** script. This script is installed when you install the Fusion Apps Development Environment extension bundle, described in Step 6 of Section 2.2.1.5, "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.

     ```
     $JDEV_MW_HOME/jdeveloper/fadev/bin/FADevInstallMwHome.py
        -m $MW_HOME
        -i /path/to/installer_files
        -w /path_to/FAStandaloneWork -v
     ```

Example 2–1 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 directory. Used for response files, unzip installers
        Defaults to the current directory
   -r : reinstall MW_HOME
   -v : verbose
```

   - **-m <mw_home>**: Use an MW_HOME 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 directory for log and other temporary files.
   - **-r**: Reinstall MW_HOME. Note that this will remove all items in the existing MW_HOME directory.
   - **-v**: Turn on verbose mode.

4. Patch MW_HOME. For more information, see the *Oracle Fusion Middleware Patching Guide*.

For the standalone domain creation to succeed, you *must* patch the `atgpf` directory in the standalone MW_HOME 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.

Follow these steps to patch the `atgpf` directory in the standalone MW_HOME using all the opatches that are in the repository:

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
```

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%

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

5. 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.

```
$JDEV_MW_HOME/jdeveloper/fadev/bin/FADevCreateDomain.py
   -m $MW_HOME
   -p $JDEV_MW_
HOME/jdeveloper/system11.1.1.5.xx.yy.zz/o.jdevimpl.rescat2/fusion_apps_
wls.properties
   -i /path/to/installer_files
   -w /path_to/FAStandaloneWork -v
```

FADevCreateDomain.py options

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

Example 2–2 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 directory for log and other temp files
-v : verbose
```

- **-p <properties file>**: Use a different **fusion_apps_wls.properties** file to configure the domain.

- **-m <mw_home>**: Use an MW_HOME 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 directory for log and other temporary files.
- **-v**: Turn on verbose mode.

2.3.1.3 Managing the Standalone WebLogic Server Lifecycle

There will be times when you want 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.

- Stop the server

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

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

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

- Remove the domain

  You may want to start over by removing the domain. Use the same xterm that was used to create the Standalone WebLogic Server domain and execute these commands:

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

- 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 Section 2.2.2, "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.

- 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.

  ```
  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/system11.1.1.5.xx.xx.xx/o.jdevimpl.rescat2/fusion_apps_
wls.properties
   -i /path/to/Repository/installers
   -w /path/to/FAStandaloneWork -v
```

2.4 Configuring Oracle SOA Suite and Oracle Enterprise Manager Fusion Middleware Control

This section discusses configuration options for the Oracle Service Oriented Architecture Suite and Oracle Enterprise Manager Fusion Middleware Control servers.

2.4.1 How to Use the Application Logging Service

> **Note:** Only Oracle SOA Suite applications developers need to perform these steps.

To use the Application Logging Service, complete these steps.

1. Set up your environment to use the Oracle SOA Suite and Java Apps Logger.
2. Update the **oracle.soa.bpel.jar**, as shown in Example 2–3.

Example 2–3 Updating the oracle.soa.bpel.jar

```
cp $MW_HOME/jdeveloper/jdev/oaext/services/Applcore-Logging-XPath.jar $MW_
HOME/jdeveloper/soa/modules/oracle.soa.ext_11.1.1
cp $MW_HOME/jdeveloper/jdev/oaext/services/Applcore-Logging-XPath.jar $MW_
HOME/jdeveloper/soa/modules/oracle.soa.ext_11.1.1
$MW_HOME/modules/org.apache.ant_1.7.0/bin/ant -f $MW_
HOME/jdeveloper/soa/modules/oracle.soa.ext_11.1.1/build.xml
```

3. Update **soa-infra-wls.ear**, as shown in Example 2–4.

Example 2–4 Updating soa-infra-wls.ear

```
pushd $MW_ORA_HOME/soa/applications
mkdir -p soa-infra-wls
cp soa-infra-wls.ear soa-infra-wls.ear.orig
cd soa-infra-wls
#add library reference
unzip -o ../soa-infra-wls.ear META-INF/weblogic-application.xml
mv META-INF/weblogic-application.xml META-INF/weblogic-application.xml.orig
sed '/<\/weblogic-application>/i<library-ref>\n
<library-name>oracle.applcore.model</library-name>\n</library-ref>\n<library-ref>\
n <library-name>Diagnostics-Engine</library-name>\n</library-ref>'
META-INF/weblogic-application.xml.orig > META-INF/weblogic-application.xml
rm META-INF/weblogic-application.xml.orig
zip -f ../soa-infra-wls.ear META-INF/weblogic-application.xml
#add resource reference
unzip -o ../soa-infra-wls.ear ejb_ob_engine_wls.jar
unzip -o ejb_ob_engine_wls.jar META-INF/ejb-jar.xml
mv META-INF/ejb-jar.xml META-INF/ejb-jar.xml.orig
sed '/<ejb-name>BPELEngineBean<\/ejb-name>/a\ <resource-ref>\n
<res-ref-name>jdbc/ApplicationDBDS</res-ref-name>\n
<res-type>javax.sql.DataSource</res-type>\n <res-auth>Container</res-auth>\n
</resource-ref>' META-INF/ejb-jar.xml.orig > META-INF/ejb-jar.xml
zip -f ejb_ob_engine_wls.jar META-INF/ejb-jar.xml
```

```
zip -f ../soa-infra-wls.ear META-INF/weblogic-application.xml
popd
```

4. Update **config.xml** if localhost access is required, as shown in Example 2–5.

Example 2–5 Updating config.xml

```
if [ -z "$(grep '<listen-address></listen-address>' $DOMAIN_
HOME/config/config.xml)" ];then
   mv $DOMAIN_HOME/config/config.xml $DOMAIN_HOME/config/config.xml.orig
   sed
's@<listen-address>.*</listen-address>@<listen-address></listen-address>@'
$DOMAIN_HOME/config/config.xml.orig > $DOMAIN_HOME/config/config.xml
fi
```

5. Restart (or start) Integrated WebLogic Server.

2.4.2 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 2–6.

> **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.
>
> ```
> alter system set processes=200 scope=SPFILE;
> ```

Example 2–6 Creating database schemas

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

2. Drop the Repository, as shown in Example 2–7. Enter the SYS password when prompted.

Example 2–7 Dropping the repository

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

   If the `-silent` switch is omitted, a wizard will be launched. It will ask you to enter the same values as shown in Example 2–6.

3. Recreate the Repository, as shown in Example 2–8. Enter the SYS password when prompted.

Example 2–8 Recreating the repository

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

```
-component ORASDPM -component BAM
```

Note: You will need to supply passwords for the different users. You should make the username and the password for that user the same, such as jmaus/jmaus. You will have to remember all the passwords. You will need them when you configure the DataSources.

2.5 Using Deployment Profiles Settings

When creating an ADF library deployment profile, you can include connection information. When a project attaches that ADF library, the connection information is merged with its own connection information. This provides runtime consistency. The 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 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 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 ADF library with the defaults. All of the connection information for that Financials workspace is included in the ADF library. HCM picks up that 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 ADF libraries, it merges the connection information into the CRM workspace; which now contains all of Financials plus HCM plus CRM.

Cleanup

Developers should audit the current deployment profiles for all of Oracle Fusion applications to make sure they are not including all of the connection information. Developers need to make sure their deployment profiles only include the connections that are truly needed directly by that project.

Developers also need to remove any unnecessary connections from the connections.xml files from each workspace. The connections.xml file should be a superset of all of these connections and not include unnecessary connections.

2.5.1 How to Use Service Deployments

A project that contains an ADF Business Components-based service can have two purposes. The ADF Business Components code can be invoked as a service or it can be used as a regular ADF Business Components object. 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 EJB JAR file that contains the actual implementation.

For use as an ADF Business Components object, consumers must get an ADF library. That is the only way the ADF Business Components objects are exposed to consumers in the ADF Business Components design time wizards. 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 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 needs to be an ADF library because consumers of this need a connection entry to be injected into the consumers' connection.xml. This does not happen with ordinary JAR files.

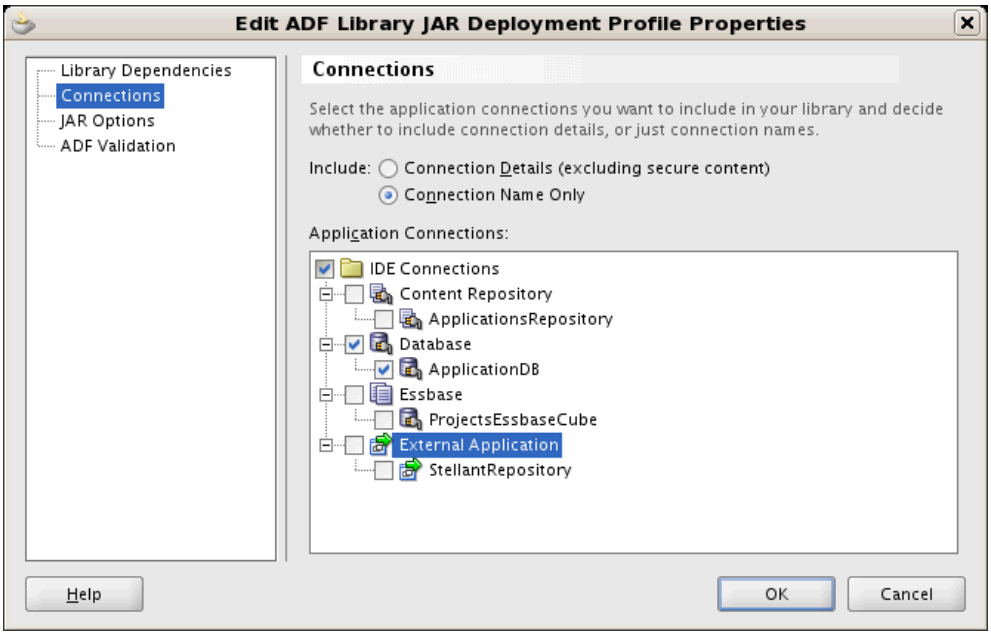
2.5.2 How to Update the Standard

All 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 2–41

Figure 2–41 Editing a Deployment Profile



2.6 Configuring the Oracle Enterprise Scheduler (ESS)

This section covers how to configure your runtime environment to support Oracle Enterprise Scheduler functionality.

For information about using the Oracle Enterprise Scheduler, see the *Oracle Fusion Applications Developer's Guide for Oracle Enterprise Scheduler*.

For information about setting up cross-domain security, see Enabling Trust Between WebLogic Server Domains.

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

2.6.1 How to Provision the Runtime Environment

Section 2.3, "Setting Up the Personal Environment for Standalone WebLogic Server" shows how to configure the **fusion_apps_wls.properties** file and run **FADevConfigDomain.py** 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 2–9 shows correctly configured Oracle Enterprise Scheduler database and schema information, and ESS-related settings.

Example 2–9 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=weblogic1
useCentralLdap=no
...
[wlsconfig]
fusionDbHost=fpp-ta02.us.oracle.com
fusionDbPort=1522
fusionDbSid=fppta02
...
# leave oraessDbHost, oraessDbPort, oraessDbSid blank if using fusion database
oraessDbHost=
oraessDbPort=
oraessDbSid=
oraessDbUser=oraess_d8b2
oraessDbPassword=oraess_d8b2
#
essMdsDbHost=
essMdsDbPort=
essMdsDbSid=
essMdsDbUser=fusion_mds_ess
essMdsDbPassword=fusion_mds_ess
```

After provisioning the runtime, you should create the supporting database schema before starting the managed servers which is covered in Section 2.6.2, "How to Create Supporting Database Schema."

2.6.2 How to Create Supporting Database Schema

There are two approaches to creating the database schema: using the RCU tool 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 RCU tool, see Section 2.4.2, "How to Use Alternate Database Schemas." Example 2–10 shows how to configure the Oracle Enterprise Scheduler schema using the RCU schema.

Example 2–10 Configuring Oracle Enterprise Scheduler Schema Using the RCU Schema

```
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
```

Alternatively, creating the schema by running scripts in SQL*Plus can be performed as shown in Example 2–11

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

Example 2–11 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
```

2.6.3 Post-Installation Checks

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

2.6.3.1 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.

2.6.3.2 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.

2.6.3.3 Checking ESS Health

Run ESS Health checks by accessing these links:

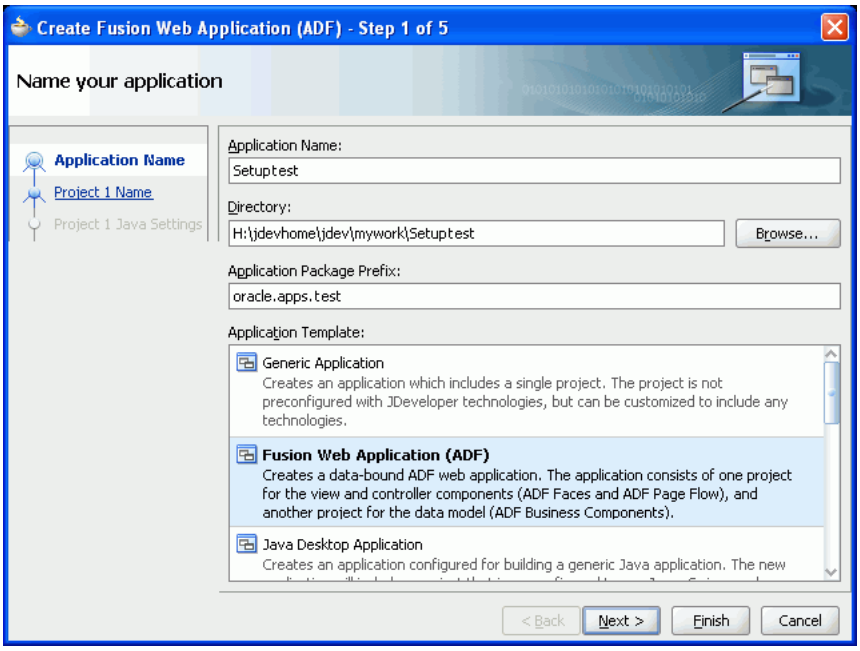
- Checking health of an ESS Node:
  `http:/<hostName>:<port>/EssHealthCheck/checkHealth.jsp`
- Checking health of an ESS Cluster:
  `http:/<hostName>:<port>/EssHealthCheck/diagnoseHealth.jsp`

2.7 Testing Your Installation

To test your JDeveloper and 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 2–42.

Figure 2–42 Naming Your Application



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*

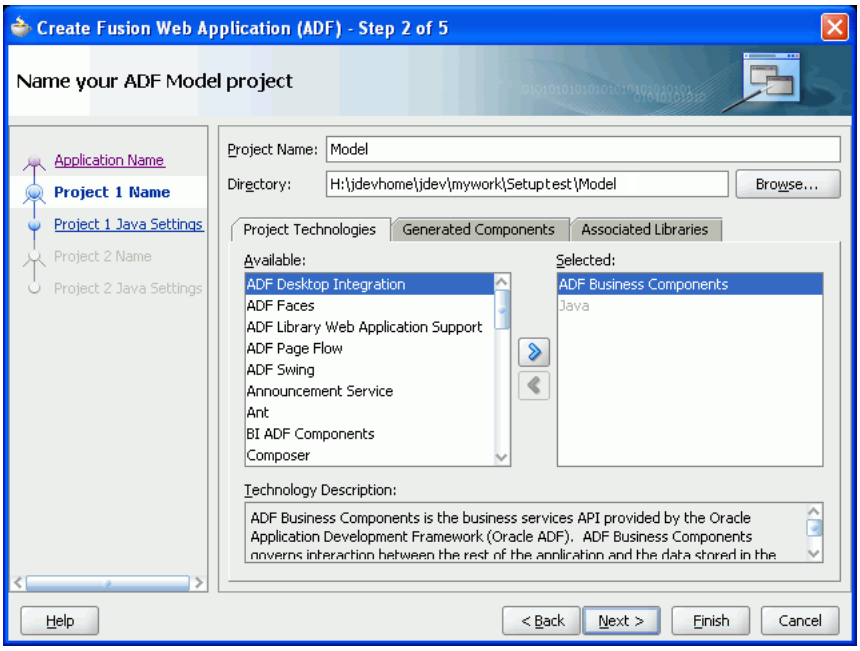
   Application Template: Choose *Fusion Web Application (ADF)*

3. Click **Next** to access the Name Your ADF-Model Project dialog. See Figure 2–43.

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 2–43 Naming Your ADF Model Project

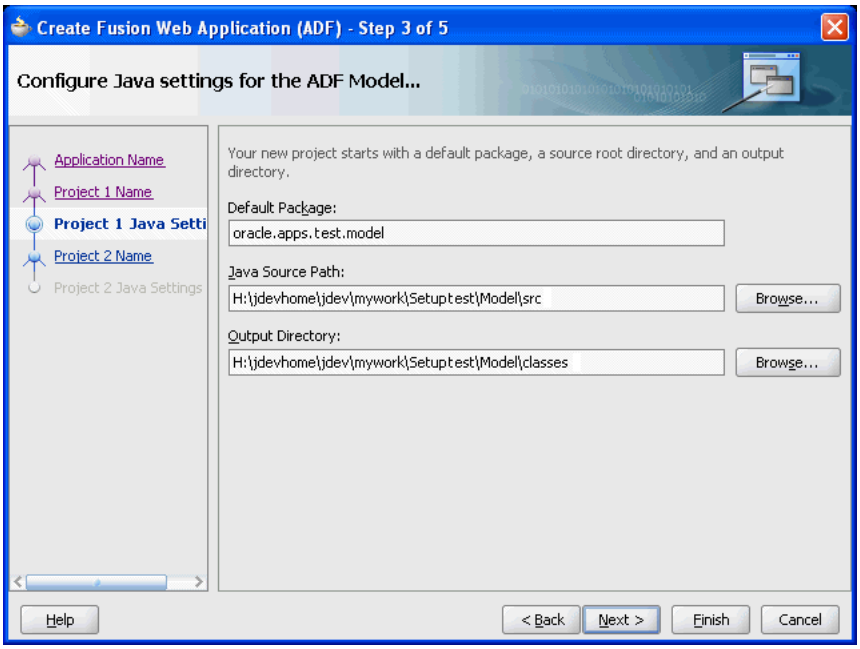


Note: The Project Technologies are automatically selected based on the application template that was chosen. You can select additional technologies if required.

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 2–44.

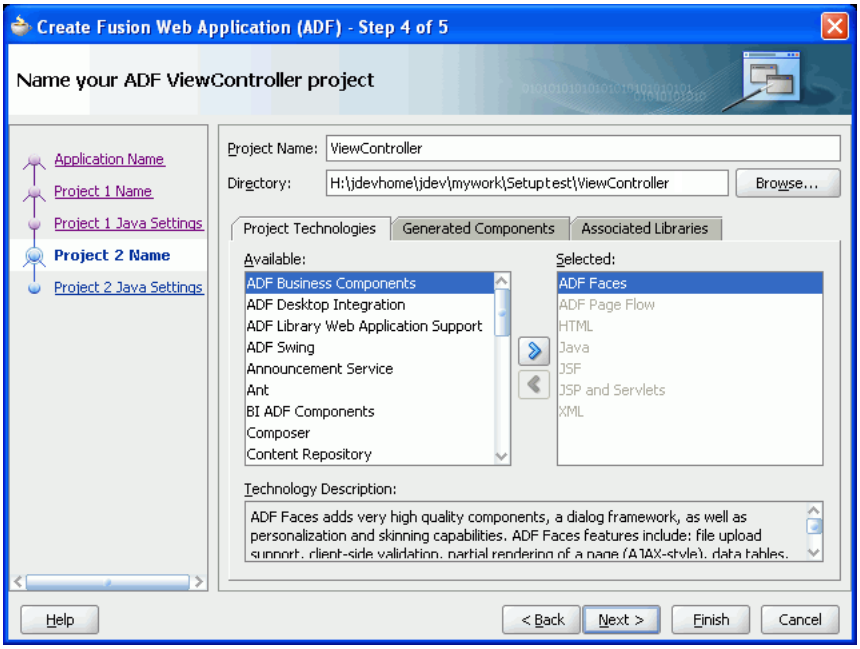
Figure 2–44 Configuring Java Settings for the ADF Model



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 2–45.

Figure 2–45 Naming Your ADF User Interface Project

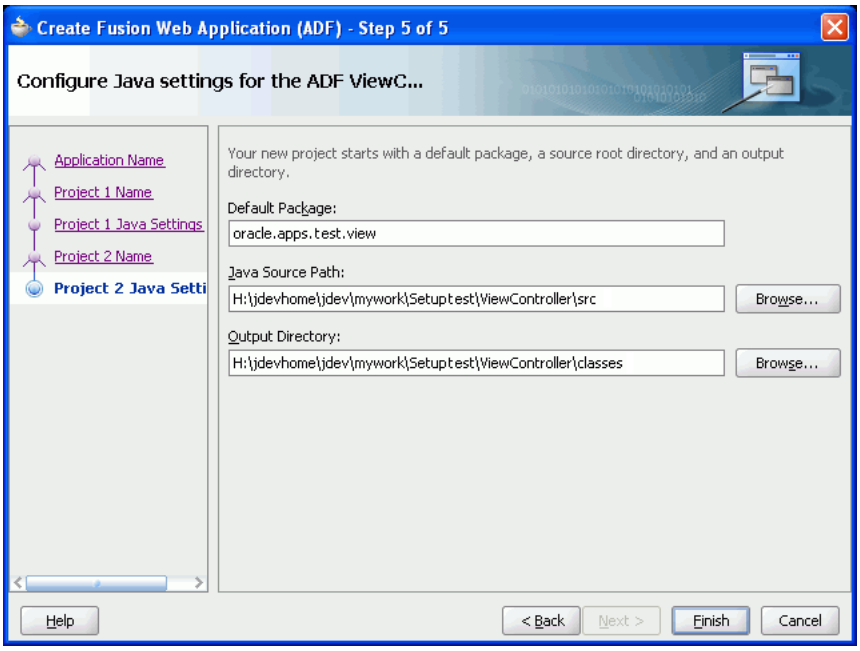


Note: The Project Technologies are automatically selected based on the application template that was chosen. You can select additional technologies if required.

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

This dialog displays the Java settings for your user interface project. See Figure 2–46.

Figure 2–46 Configuring Java Settings for the ADF User Interface Project



Click **Finish** to create your new application.

7. Add the *Applications Core*, *Applications Core (Attachments Model)*, and *Java EE 1.5* libraries to the data model project. See Section 3.3, "Adding the Applications Core Library to Your Data Model Project."

8. Add the *Applications Core (ViewController)* tag library to the user interface project. See Section 3.4, "Adding the Applications Core Tag Library to Your User Interface Project."

9. Create the *ApplicationDB* database connection for your team's database. See Section 3.6, "Creating a Database Connection." Enter these connection details:

   Connection Type: Choose *Oracle (JDBC)*

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

   Deploy Password: Select this option.

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

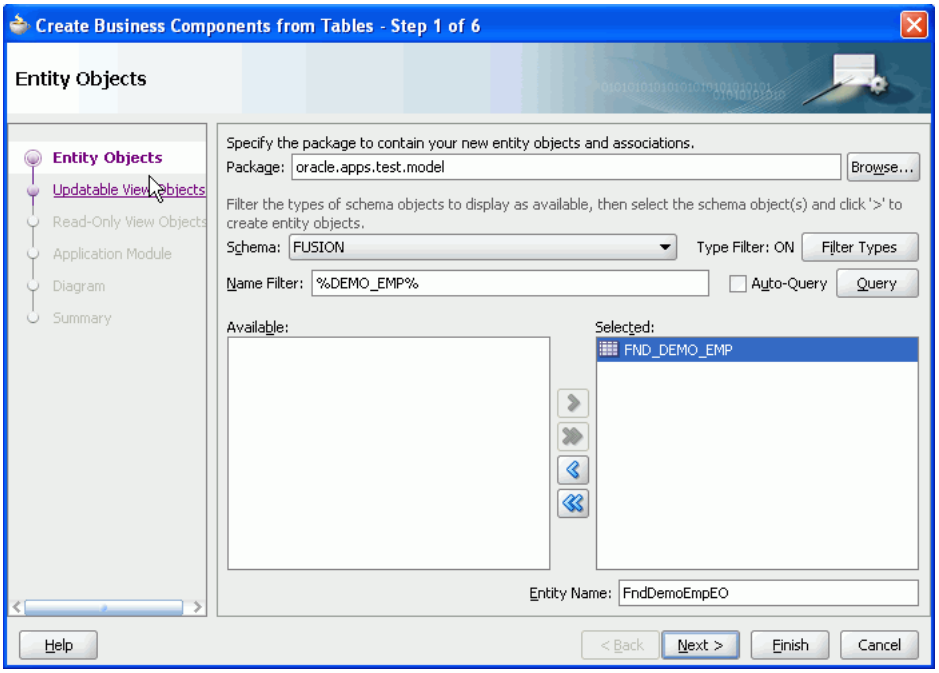
   JDBC Port: Enter the port number for your database.

   SID: Enter the database name, such as mydb.

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

11. 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 2–47.

Figure 2–47 Create Business Components from Tables — Entity Objects

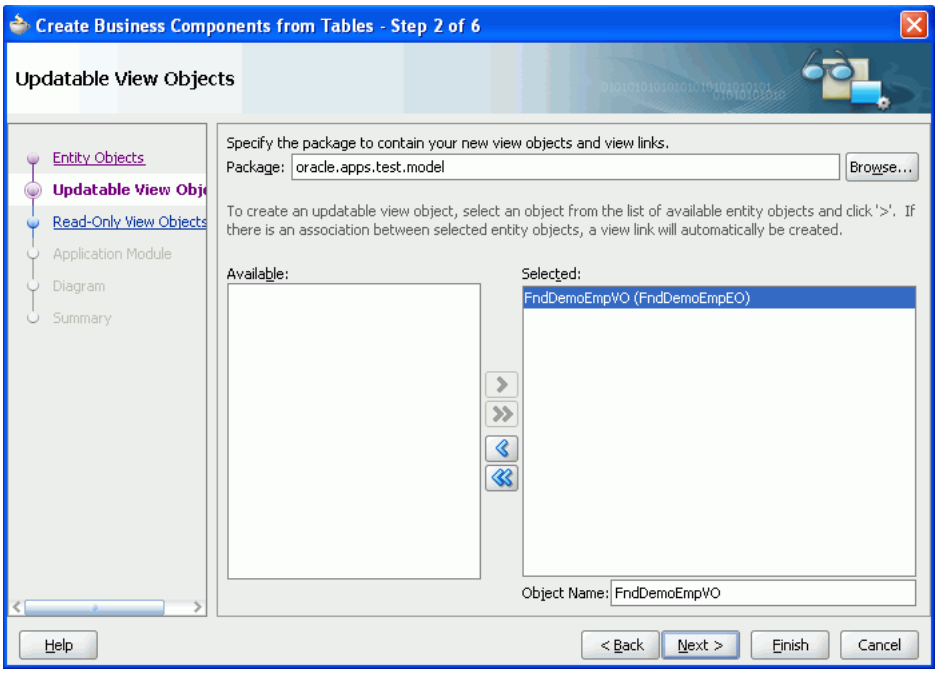


12. Complete the following to create your entity object:
    a. **Filter Types**: Select only *Tables* to narrow your search for schema objects.
    b. **Filter Name**: Enter a filter, such as *%DEMO_EMP%* to narrow your search to tables.
    c. **Query**: Click this button to perform your search.

    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.
13. Choose the required entity object located in the **Available** column and click **>** to shuttle it over to the **Selected** column. See Figure 2–48.

Figure 2–48 Create Business Components from Tables — Updatable View Objects



14. Click **Finish** to create your updateable view object and to close the Business Objects wizard.

15. 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 **Application Navigator** and select your application module. Right-click and select **Configurations** from the menu.

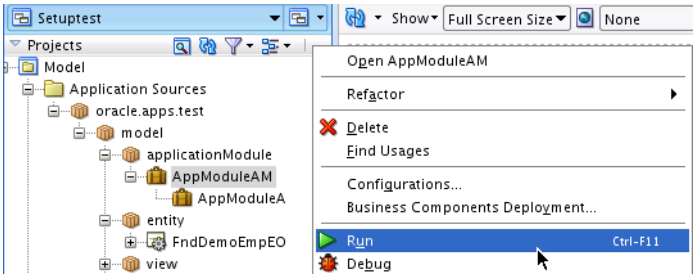
    b. Choose the configuration *<AM Name>Local*, then choose **Edit**.

    c. Change the **Connection Type** to *JDBC DataSource* and **Datasource Name** to *java:comp/env/jdbc/ApplicationDBDS*. Click **OK**.

16. 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.

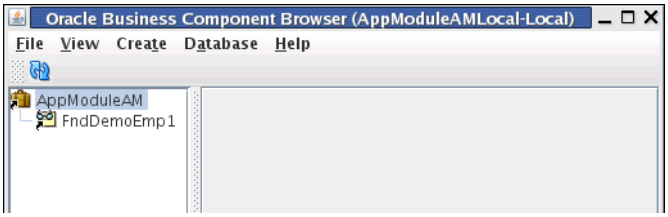
    In the Navigator tree, right-click the application module and select **Run**, as shown in Figure 2–49.

Figure 2–49 Running the Application Module



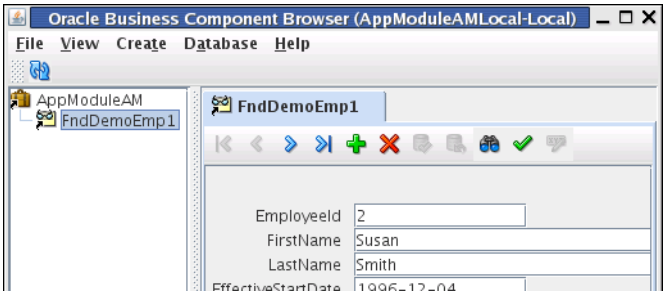
If your installation is set up correctly, a dialog similar to that shown in Figure 2–50 displays. If an error message displays, you will need to re-check that the previous steps have been performed correctly.

Figure 2–50 Application Module in Business Component Browser



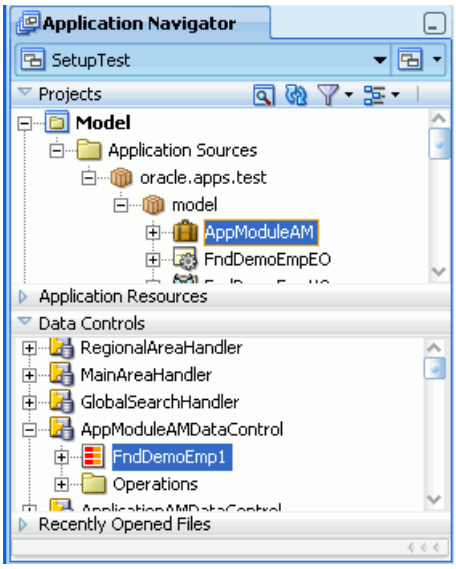
In this example, right-click FndDemoEmp1 and select **Show** to display data, as shown in Figure 2–51.

Figure 2–51 Showing Data in the Business Component Browser



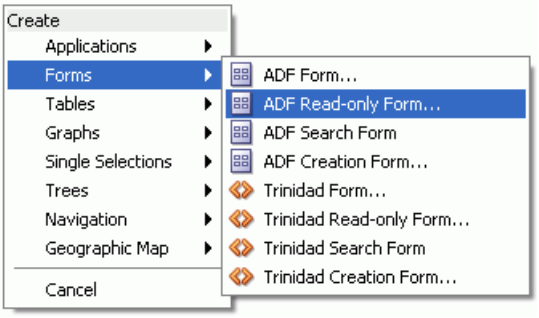
17. Choose **Application Navigator > ViewController**. Right-click and select **New** from the menu to open the New Gallery.
18. Choose the **Web Tier > JSF** category. Select the **JSF Page** item and click **OK** to open the Create JSF Page dialog.
19. Complete the following:
    - **File Name**: Enter *Setup.jsp*
    - Select to create jspx file.
    - Click **OK**.
20. Go to the Data Controls panel and drag the collection onto the open Setup.jspx window. See Figure 2–52.

Figure 2–52 Data Controls Panel - FndDemoEmp1 Collection



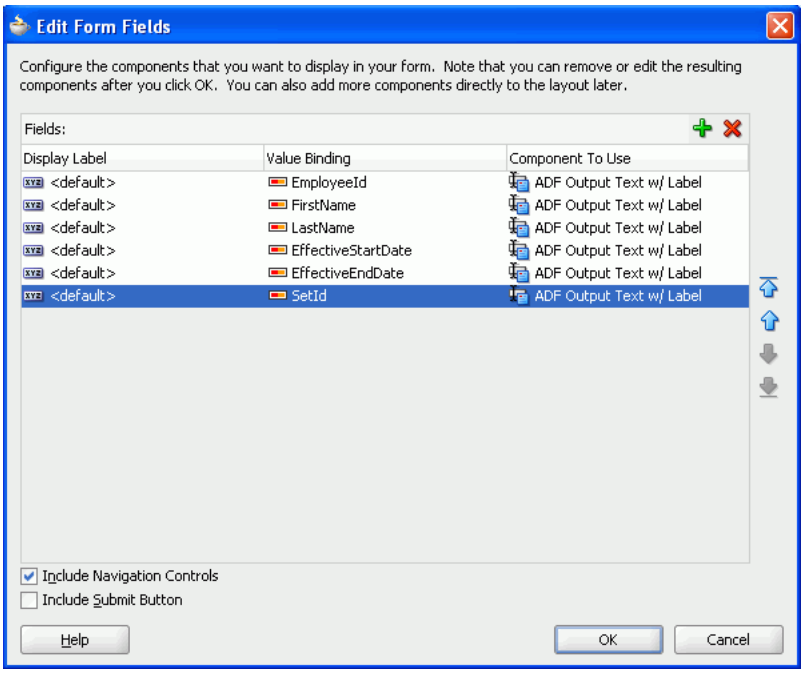
21. Select to create **Forms > ADF read-only** from the context menu that displays. See Figure 2–53.

Figure 2–53 Context Menu



22. 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 2–54.

Figure 2–54 Editing Form Fields



23. 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/ApplCoreCRMDemo/faces/Region6UIShellPage`, you should edit it manually so it appears similar to `http://myhost.name.com:7101/ApplCoreCRMDemo/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.

2.8 Using Best Practices for Setting Up the Development Environment

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

2.8.1 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 **jdev.conf** because it will just result in duplicating the parameter on the command line because it already is set in **ide.conf**. 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 $*jdev_install*/jdev/bin/jdev.conf.

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

The heap monitor shows the current size of the heap; not necessarily the maximum size. The heap is originally 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.

2.8.2 How to Refresh the ADF Library Dependencies Library

The 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 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 ADF Library Dependencies should be used to update the file. This could happen because of updated lower-level dependency changes outside of the design time session.

2.8.3 How to Create the Integrated WebLogic Server Domain

The domain for Integrated WebLogic Server is generated from the information in **fusion_apps_wls.properties**. This file is created by the wizard described in Section 2.2.2, "How to Use the Oracle Fusion Domain Wizard."

If you do not have a valid **fusion_apps_wls.properties** in the extension directory, domain creation probably will fail. So, if your domain creation fails, the **fusion_apps_wls.properties** file should be the first thing you check.

The **fusion_apps_wls.properties** file also is used for Standalone WebLogic Server.

Some properties may become a source of problems if not configured properly.

useCentralLdap: This property should be set to Yes if the environment needs to use the central LDAP.

The new **fusion_apps_wls.properties** will not take effect until the Integrated WebLogic Server domain is recreated. The domain is created automatically when you run Integrated WebLogic Server from JDeveloper for the first time in a new view.

The domain is created under your JDeveloper system directory, so that each view has its own domain.

```
$JDEV_USER_HOME/system11.1.1.2.37.55.96/DefaultDomain/...
```

If you delete the **DefaultDomain** directory from under the system directory, the next time you start Integrated WebLogic Server, it will be recreated from the latest **fusion_apps_wls.properties**.

2.8.4 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. Session servers and workstations may respond differently.

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.

Once 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.

```
$ "/usr/sbin/lsof -i -P | grep 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>
```

2.8.5 How to Work with ADF Libraries at Design Time

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

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 ADF libraries get managed under a Library called **ADF Libraries**. Mixing and matching different methods of adding 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 ADF libraries are resolved when the workspace is loaded in JDeveloper. If a reference to a component or Java class in an 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 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 ADF libraries.

When you make any changes to the components in a project, where the components are being referenced as an ADF library by your user interface project, you need to redeploy the ADF library and refresh the 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 ADF library / refreshing ADF library references each time you make a change.

2.9 Configuring Hierarchy Providers for Approval Management (AMX)

Human Capital Management (HCM) maintains complex hierarchies and uses web services to retrieve this information. These services are known as service extensions. One of these extensions is the hierarchy provider, which allows you to walk up a hierarchy to retrieve information about a manager or subordinate. A simple example would be you, your manager, your manager's manager, and so on.

See "Using Approval Management" in *Oracle Fusion Middleware Modeling and Implementation Guide for Oracle Business Process Management*.

Before you begin

Before you can configure hierarchy providers, you need to update the credential store using the WebLogic Scripting Tool. Follow these steps:

1. Run **wlst.sh** from the current working directory and answer the prompts.
   - wls:/offline> connect()
   - Please enter your username [weblogic]:weblogic
   - Please enter your password [weblogic]:weblogic1
   - Please enter your server URL [t3://localhost:7001]:t3://999.99.999.99:7101
   - Connecting to

     t3://localhost:7101 with userid weblogic ...
2. Run these WLST commands (exactly as they are here) to create credentials within the domain's credential store.

```
wls:/DefaultDomain/serverConfig> updateCred(map="oracle.wsm.security",
key="keystore-csf-key", user="owsm", password="welcome1", desc="Keystore key")
wls:/DefaultDomain/serverConfig> updateCred(map="oracle.wsm.security",
key="enc-csf-key", user="orakey", password="welcome1", desc="Encryption key")
wls:/DefaultDomain/serverConfig> updateCred(map="oracle.wsm.security",
key="sign-csf-key", user="orakey", password="welcome1", desc="Signing key")
wls:/DefaultDomain/serverConfig> updateCred(map="oracle.wsm.security",
key="basic.credentials",user="weblogic",password="weblogic",desc="User
credentials key")
wls:/fusion_domain/serverConfig>  createCred(map="oracle.wsm.security", key="
FUSION_APPS_AMX_APPID-KEY ", user="FUSION_APPS_AMX_APPID", password="Welcome1",
desc="User credentials key")
```

   After running the commands, a message similar to this displays:

```
desc=User credentials key, map=oracle.wsm.security, password=Welcome1,
user=FUSION_APPS_AMX_APPID, key=FUSION_APPS_AMX_APPID-KEY }
```

There are three types of hierarchy providers.

- **Supervisory**: A hierarchy that allows you to walk up and down a user's management chain.
- **Job-level**: A supervisory hierarchy provider with additional job-level information attached to each user.
- **Position**: An HCM hierarchy in which a position's manager is another position. Position has one or more users as its members. Because position is not a native entity in Identity Service Management, you need to set up additional web services to retrieve its data. These services are considered to be service extensions. Service extensions include hierarchy-provider, position-lookup, and display-name-lookup web services.

Each list builder may have a corresponding hierarchy provider.

A hierarchy principal is something that participates in the hierarchy. It has certain parameters that the hierarchy provider uses to determine which hierarchy to walk up to. These parameters are:

- userID
- assignmentID
- effectiveDate (of the assignment)
- hierarchyType (to use)
- propertyBag (additional parameters map)

> **Note:** Integration with Oracle HCM is native. That is, you provide the WSDL URL for each hierarchy provider.

Service extension is defined in the `workflow-identity-config.xml` file under `$ORACLE_HOME/user_projects/domains/soainfra/config/soa-infra/configuration`.

The file location also may come from MetaData Services (MDS). The file could be updated at the MDS location by using a WebLogic Scripting Tool command, such as importMetadata and exportmetaData. For example:

```
# ./wlst.sh
# connect('weblogic','weblogic1','t3://<your_host>:7001')
# importMetadata(application='soa-infra',server='AdminServer',fromLocation='<your_
path>',docs='/soa/configuration/default/workflow-identity-config.xml')
# exit()
```

- Identity Service Configuration (tag ISConfiguration) has two sections: configurations and service extensions.
- Under configurations, service extension is specified using the IdentityServiceExtension property.
- Under service extensions (tag serviceExtensions), three hierarchy providers can be specified: JobLevel hierarchy provider, Supervisory hierarchy provider, and Position hierarchy provider. Two special service providers can be specified: position-lookup and position-display-name. All are service providers. JobLevel and Supervisory use the same Java class `oracle.bpel.services.identity.hierarchy.providers.hcm.HCMHierarchyProvider`.

The position-lookup provider allows you to look up the members of a position and all the positions that belong to a user. The position-display-name provider allows you to retrieve the display names of a list of positions for a particular language.

The sample Identity Service Configuration XML code shown in Example 2–12 specifies a service extension, HCMIdentityServiceExtention, for JpsProvider. It then specifies the providers in the service extension.

Note: Within each provider, the attribute `classname` points to a Java implementation of the service provider. The parameter `wsdlURL` points to the URL of the concrete Web Service Description Language (WSDL) for the provider's web service. You should replace this value with the actual URL.

Example 2–12 Sample workflow-identity-config.xml File for Specifying HCM Providers

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<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" />
          <property name="IdentityServiceExtension" value="HCMIdentityServiceExtension"/>
        </provider>
      </configuration>
  </configurations>
  <property name="caseSensitive" value="false"/>
<serviceExtensions>
  <serviceExtension name="HCMIdentityServiceExtension">
      <serviceProvider type="supervisoryHierarchyProvider"
classname="oracle.bpel.services.identity.hierarchy.providers.hcm.HCMHierarchyProvider">
        <initializationParameter name="wsdlUrl" value="HierarchyProviderService?WSDL"/>
         <!-- Optional parameters.  Depicts how the defaults could be overridden to specify
different values -->
        <initializationParameter name="policyFile" value="file://hcm-client-policy.xml"/> <!--
using different owsm policy, see below for sample policy file -->
        <initializationParameter name="csf-key-name" value="hcm-csf-key-other"/> <!-- using
different name for the csf-key-->
        <initializationParameter name="http-read-timeout" value="6000"/> <!-- Use this value to
specify HTTP read timeout in miliseconds, default is 5000 milisec-->
        <!-- securityPolicyName controls the local policy attachment to use.  This value is used
along with csf-key-name to use elevated privileges See Bug 10368000 for more information-->
        <initializationParameter name="securityPolicyName" value="oracle/wss10_saml_token_client_
policy"/>
      </serviceProvider>
      <serviceProvider type="positionHierarchyProvider"
classname="oracle.bpel.services.identity.hierarchy.providers.hcm.HCMPositionHierarchyProvider">
         <initializationParameter name="wsdlUrl"
value="http://<host>/HierarchyProviderService?WSDL"/>
       </serviceProvider>
      <serviceProvider type="positionLookupProvider"
classname="oracle.bpel.services.identity.position.provider.hcm.PositionLookupServiceProvider">
         <initializationParameter name="wsdlUrl"
value="http://f<host>/positionLookupService?WSDL"/>
       </serviceProvider>
      <serviceProvider type="positionDisplayNameProvider"
classname="oracle.bpel.services.identity.position.provider.hcm.PositionDisplayNameProvider">
```

```
         <initializationParameter name="wsdlUrl"
value="http://<host>/HierarchyProviderService?WSDL"/>
       </serviceProvider>
      <serviceProvider type="jobLevelHierarchyProvider"
classname="oracle.bpel.services.identity.hierarchy.providers.hcm.HCMHierarchyProvider">
         <initializationParameter name="wsdlUrl"
value="http://<host>/HierarchyProviderService?WSDL"/>
       </serviceProvider>
   </serviceExtension>
 </serviceExtensions>
</ISConfiguration>
```

3

Setting Up Your JDeveloper Workspace and Projects

This chapter describes how to set up your JDeveloper 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:

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 Workspace

The application's Enterprise Archive (EAR) will be available for developers to pick up when creating a custom workspace. 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 JDeveloper for Customizations" in *Oracle Fusion Applications Extensibility Guide*.

3.3 Adding the Applications Core Library to Your Data Model Project

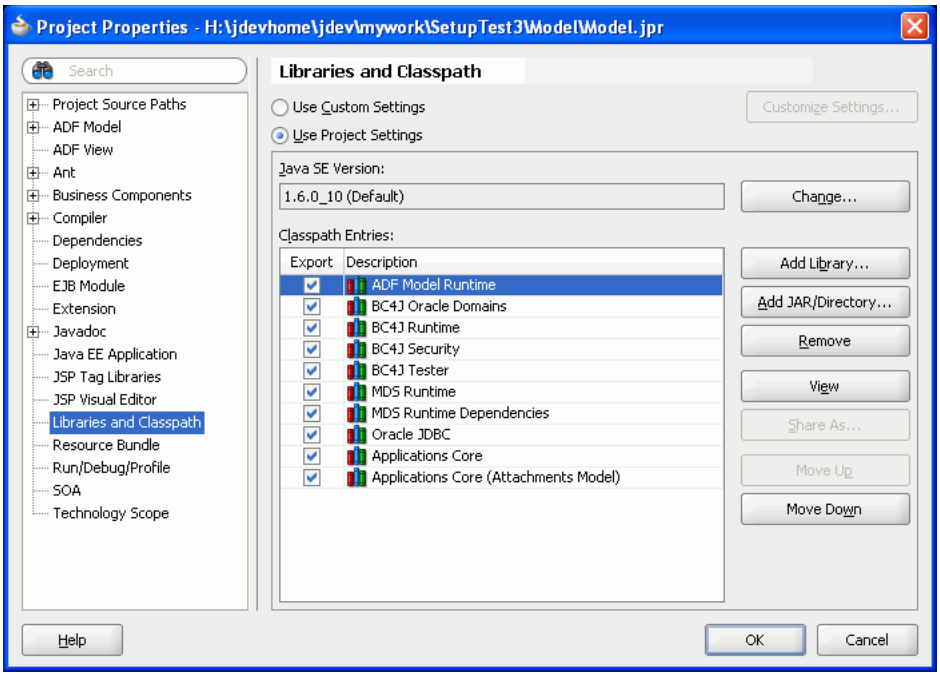
Use these directions to add the *Applications Core* and *Applications Core (Attachments Model)* libraries to the data model project. The default name, provided by JDeveloper, for this project is `Model`.

To add the Applications Core and Applications Core (Attachments Model) 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* and *Applications Core (Attachments Model)* 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 in Figure 3–1.

Figure 3–1 Project Properties — Libraries and Classpath Dialog



4. Click **OK** to save your changes.

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

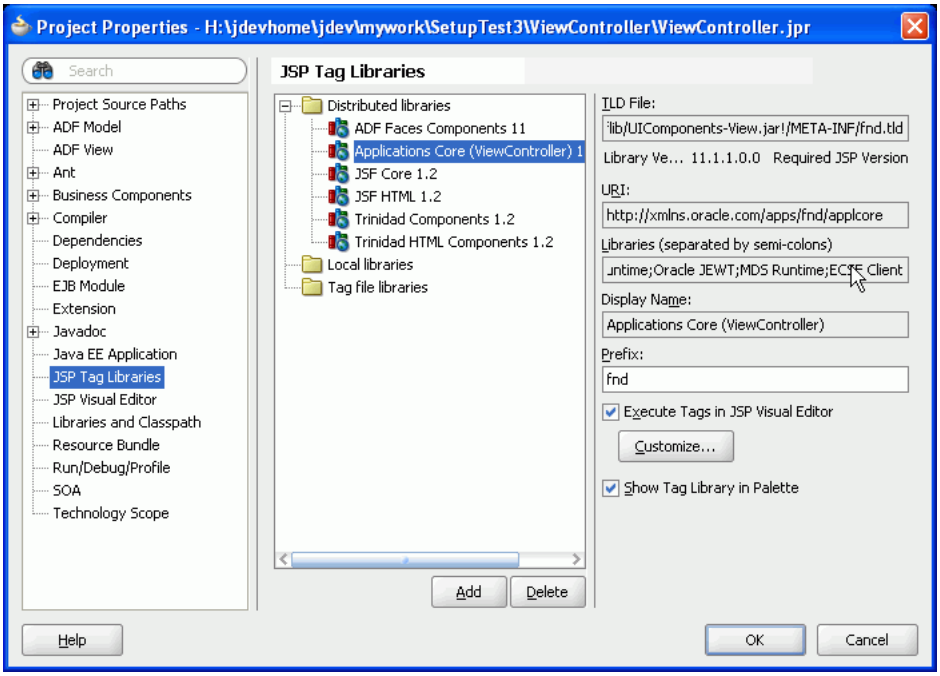
Use these directions to add the *Applications Core Tag Library* to the user interface project. The default name provided by JDeveloper for this project is `ViewController`.

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.

Figure 3–2 Project Properties — JSP Tag Libraries Dialog



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 J2EE application. Applications Core setup UIs are part of the Applications Core (Setup UI) shared library, which is hosted centrally in the Applications Core Setup J2EE application. As a result, product teams typically will not need to include the shared library in their own J2EE 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 Resource application can register setup activities such as "Create Employees" and "Manage Employee Tree Structure." See *Oracle Fusion Applications Information Technology Management, Implement Applications 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:

- Section 7.11, "Integrating Messages Task Flows into Oracle Fusion Functional Setup Manager"
- Section 8.3, "Integrating SetID Task Flows into Oracle Fusion Functional Setup Manager"
- Section 10.6, "Integrating Lookups Task Flows into Oracle Fusion Functional Setup Manager"
- Section 11.7, "Integrating Document Sequence Task Flows into Oracle Fusion Functional Setup Manager"
- Section 18.6, "Integrating Attachments Task Flows into Oracle Fusion Functional Setup Manager"
- Section 25.5, "Integrating Flexfield Task Flows into Oracle Fusion Functional Setup Manager"
- Section 54.2, "Integrating Profiles Task Flows into Oracle Fusion Functional Setup Manager"

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 product team's requirements, familiarize yourself with these three scenarios and decide which one best fits your team's 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 taskflow running in the Applications Core Setup J2EE 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 taskflow running in the Applications Core Setup J2EE 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 taskflow running in the product team's own J2EE 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 J2EE application. For example, 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

Step	Scenario 1	Scenario 2	Scenario 3
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 Section 3.5.1, "What You May Need to Know About Setup UIs in Oracle Fusion Functional Setup Manager") that describes each task flow and its parameter values.		X	X
Product teams should set the value of the Enterprise Application field (in the Application Design Repository) to the appropriate J2EE 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 J2EE application.		X	X
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.	X	X	X
If you intend to integrate a product-team specific setup UI and it will run in your product team's own J2EE application, your application will need to include the Applications Core shared library.			X
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.			X

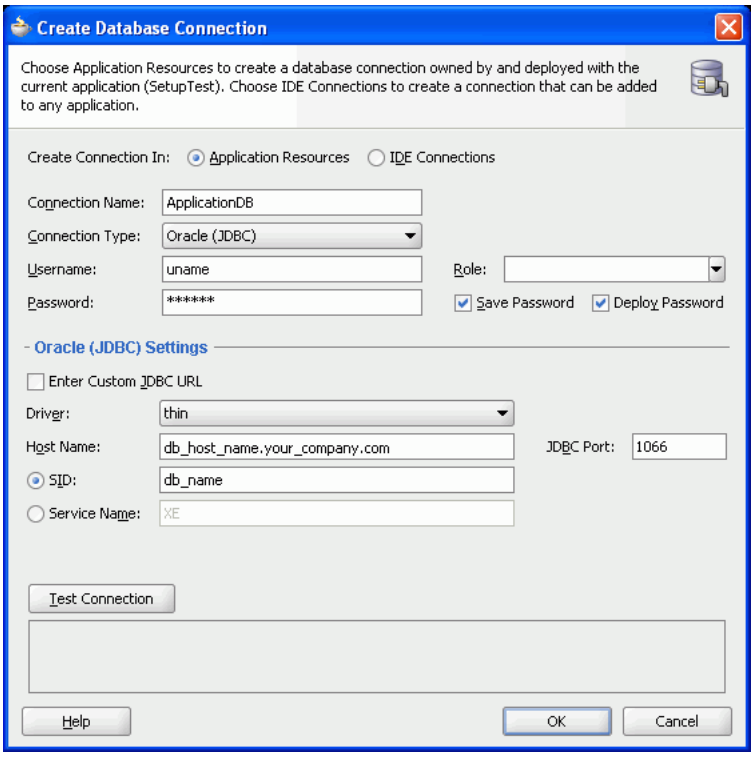
3.6 Creating a Database Connection

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

To create a database connection:

1. Choose **Application Resources > Connections**. Right-click and choose **New Connection > Database** from the menu.
2. Add 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.

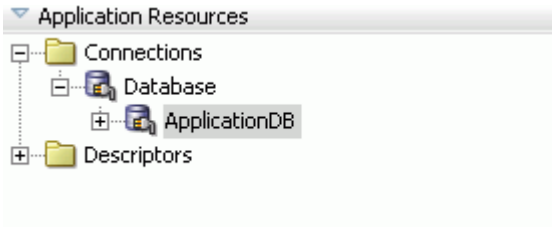
Host Name: This is the default host name if the database is on the same machine as JDeveloper. If the database is on another machine, type the name (or IP address) of the computer where the database is located.

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, a 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.

Figure 3–4 Application Resources — Connections



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.

For more information about ECSF, see Part V, "Using Oracle Enterprise Crawl and Search Framework"

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 Chapter 27, "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

Once 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 27–3.

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

For more information, see Chapter 27, "Creating Searchable Objects."

3.8 Deploying Oracle SOA Suite

For information about deploying SOA, see:

- "Deploying an SOA Composite Application" in the "Developing SOA Composite Applications with Oracle SOA Suite" chapter of the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.
- *Oracle Fusion Middleware Enterprise Deployment Guide for Oracle SOA Suite*.

3.9 Implementing Oracle Enterprise Scheduler Service Workspace and Deployment

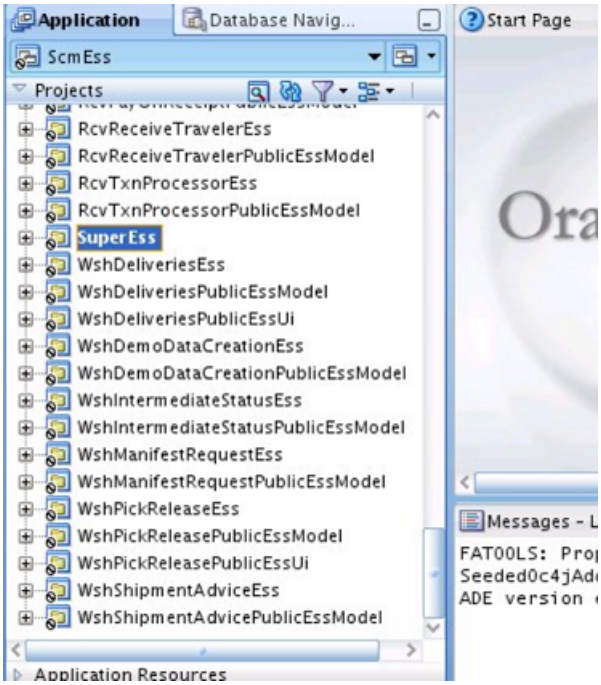
Oracle Enterprise Scheduler Service applications, also known as Oracle Enterprise Scheduler Service hosting applications, are deployed to an Oracle Enterprise Scheduler Service-configured runtime/cluster that has been pre-deployed with the base ESSAPP infrastructure J2EE application. Following standards, 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/UI task flow projects for development-time testing.
- SuperEss consolidating the Enterprise JavaBeans deployment descriptors for the entire Oracle Enterprise Scheduler Service hosting application.

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/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. 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.

6. Create the `ejb-jar.xml` and `weblogic-ejb.jar.xml` files. See "Assembling the Scheduler Sample Application" in the Oracle Fusion Applications Developer's Guide for Oracle Enterprise Scheduler. Save the files in the src/META-INF directory.

After completing these steps, the SuperEss project will be complete. Follow Section 3.9.2, "How to Build the EAR/MAR Profiles" to build the EAR/MAR deployment profiles.

3.9.2 How to Build the EAR/MAR Profiles

Oracle Enterprise Scheduler Service-hosted applications are built into EAR files and deployed as J2EE 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 via 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, having project-level deployment artifacts is essential. To support this requirement, EARs with multiple metadata archive (MAR) files can be deployed. This section describes what needs to be done to properly build Oracle Enterprise Scheduler Service workspaces to support project-level MARs.

3.9.2.1.1 How to Enable Your 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 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 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 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 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 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**.

   f. Click **Add** to add a new contributor and browse to and select the **essmeta** project-level directory.

   g. Select **File Groups > Project Output > Filters** and confirm the addition of the relevant Oracle Enterprise Scheduler Service metadata files.

   h. Click **OK**.

   i. Deploy as JAR and verify.

3. Complete the 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 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 Building 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. Under no circumstances should the Oracle Enterprise Scheduler Service hosting application's EAR file 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 Deploying 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.

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. Once 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 should 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**.

Once 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 Oracle Fusion Applications Developer's Guide for Oracle Enterprise Scheduler. This completes the deployment discussion for Oracle Enterprise Scheduler Service hosting applications.

3.10 Implementing Oracle Application Development Framework UI Workspace and Projects

Before you can actually deploy your web project, there are a number of preliminary steps that you need to accomplish. 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 are automatically defaulted for you. However, there are still certain options that you need to manually set 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 this project that JDeveloper provides is `ViewController`.

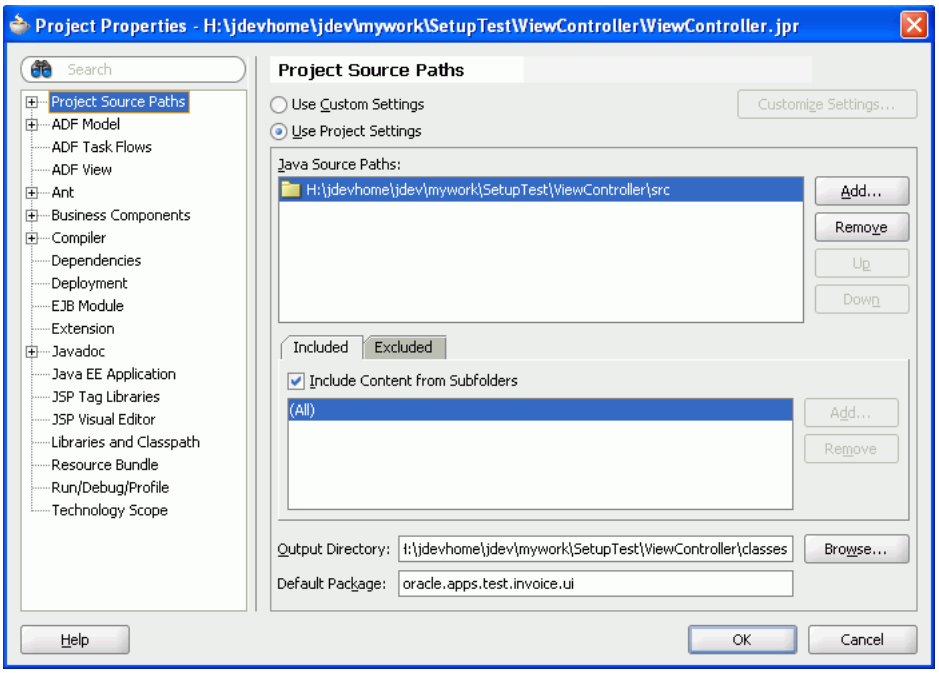
3.10.1.1 Configuring Your User Interface Project

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.

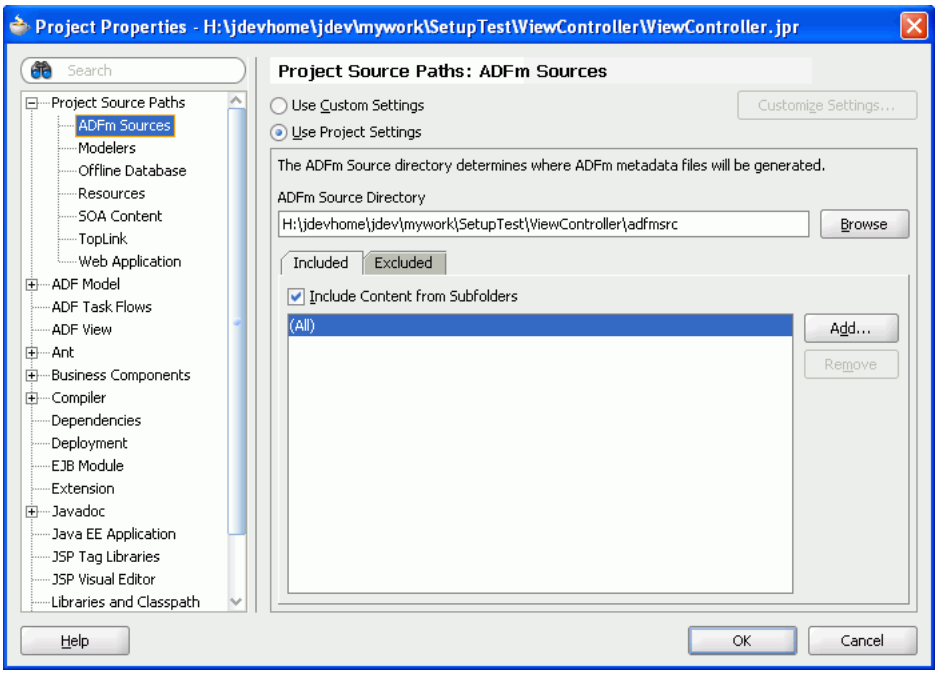
Figure 3–6 Project Properties — Project Source Paths Dialog



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 Path: ADFm Sources dialog, as shown in Figure 3–7.

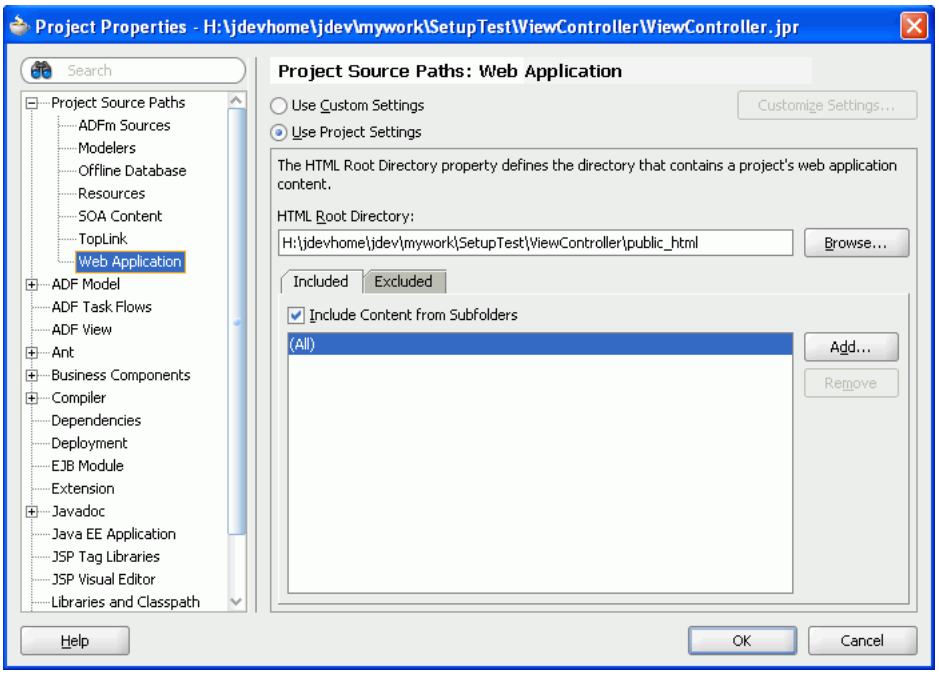
Figure 3–7 Project Properties — Project Source Paths: ADFm Sources Dialog



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 Path: Web Application dialog, as shown in Figure 3–8.

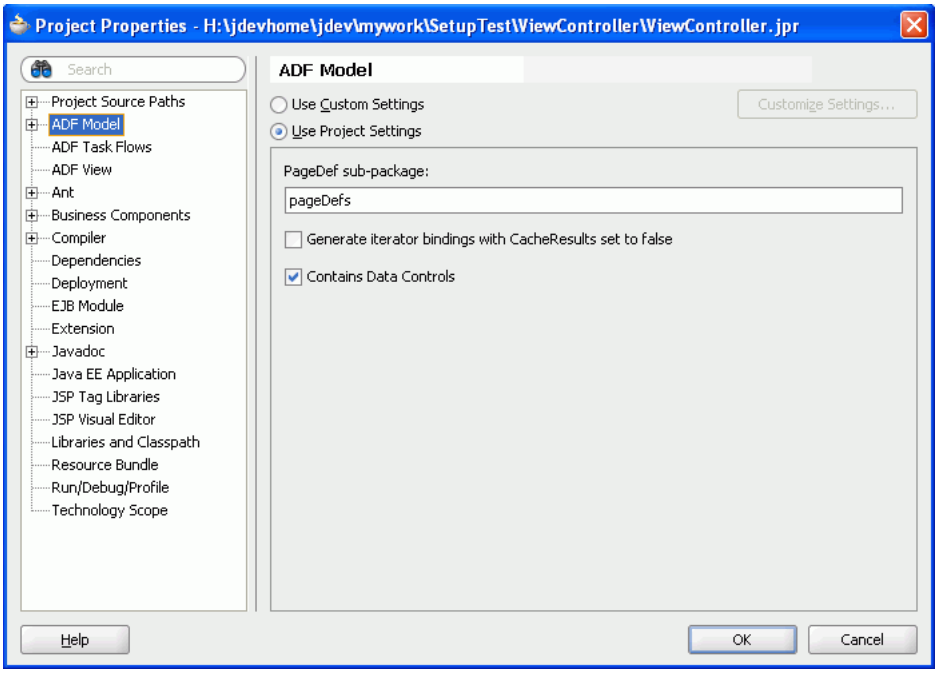
Figure 3–8 Project Properties — Project Source Paths: Web Application Dialog



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.

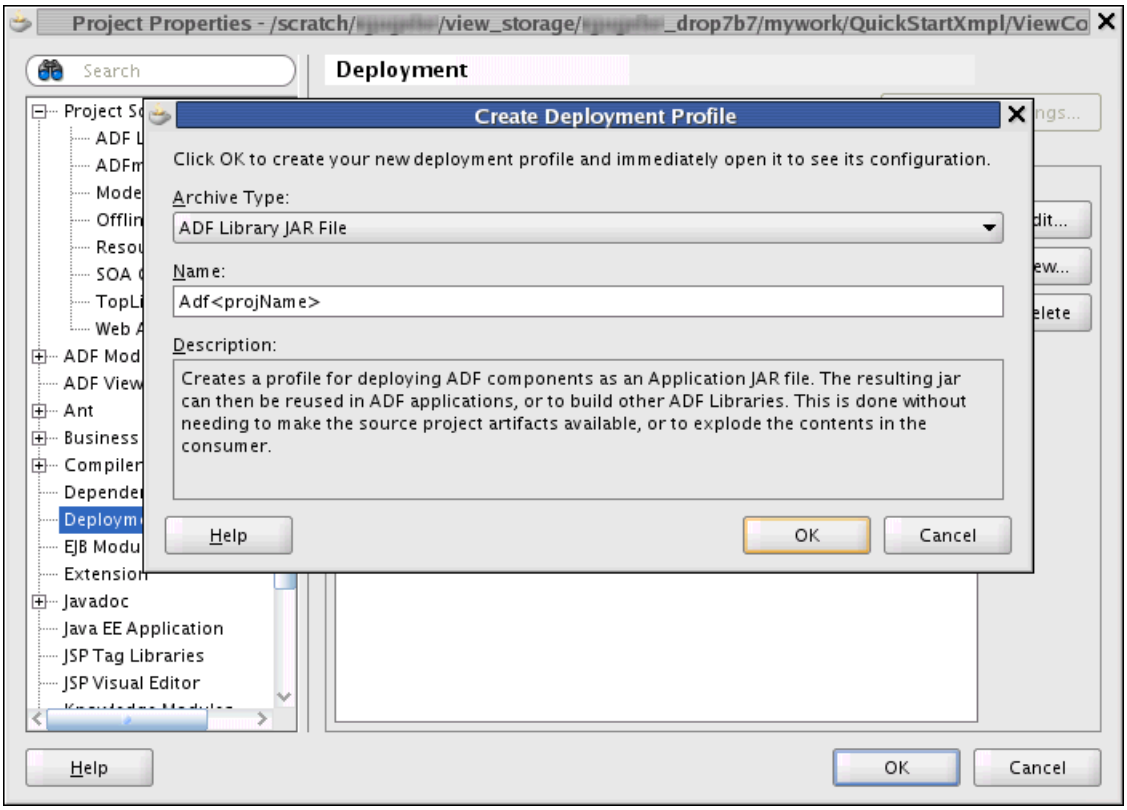
Figure 3–9 Project Properties — ADF Model Dialog



The location of the page definition files are 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.

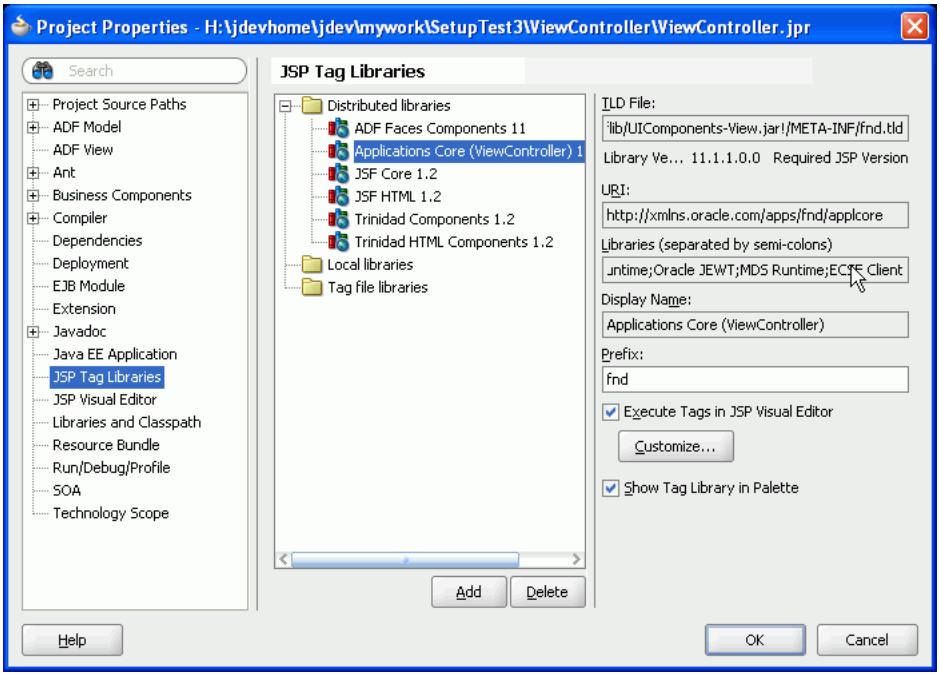
   Enter the **Name** as `Adf<projName>` in accordance with the Package Structure and Naming Standards.

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.

Figure 3–11 Project Properties — JSP Tag Libraries Dialog



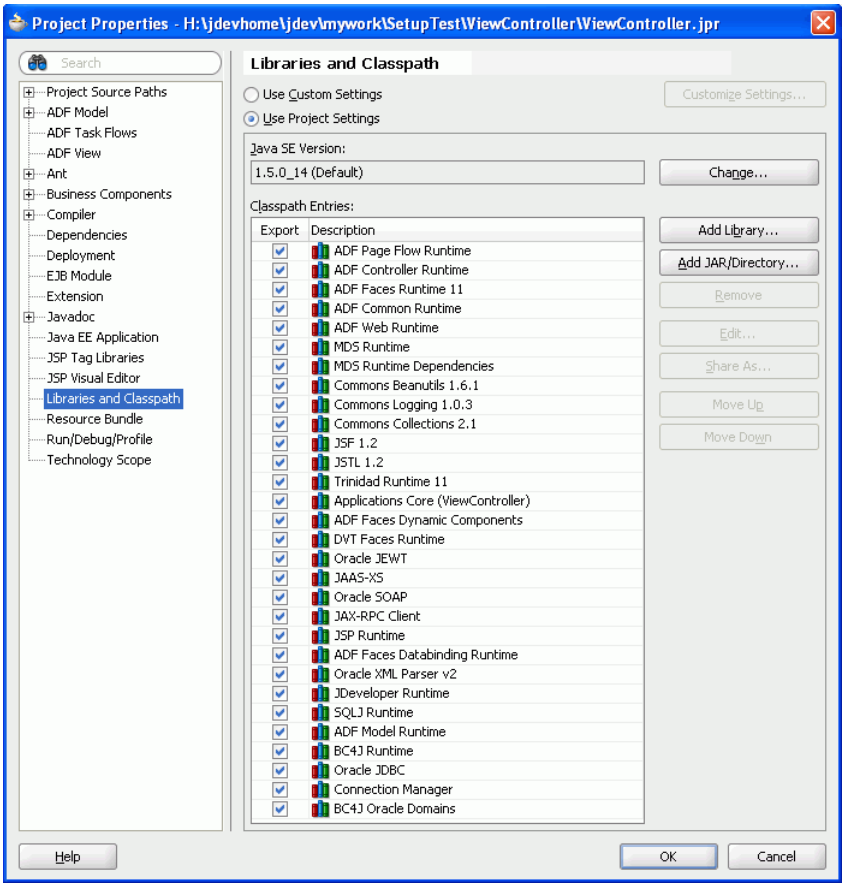
Verify that you have the following tag libraries listed under the **Distributed libraries** folder:

- 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 Section 3.4, "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.

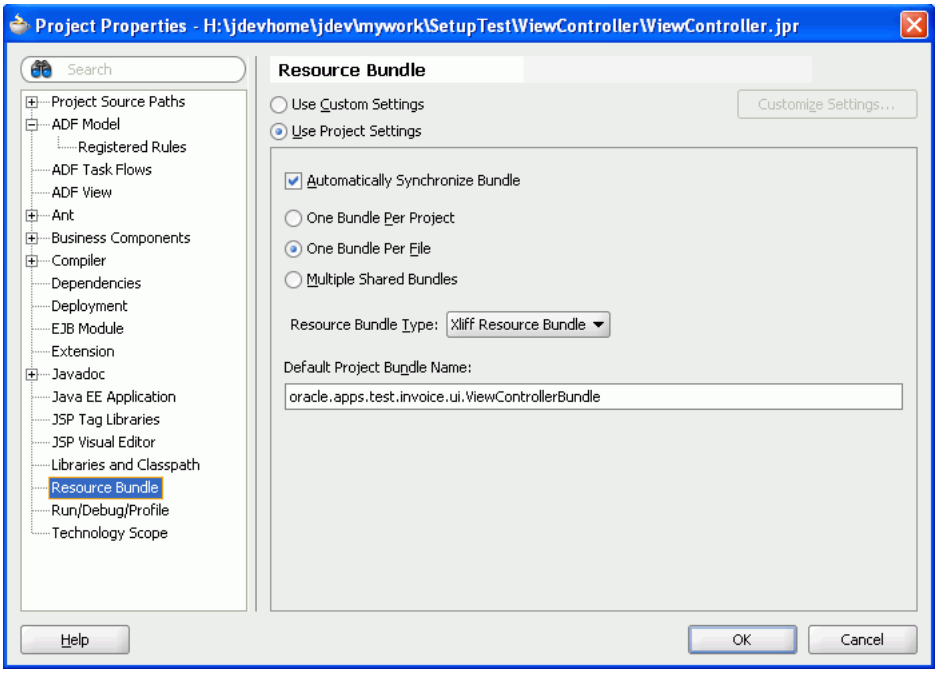
Figure 3–12 Project Properties — Libraries and Classpath Dialog



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.

13. Choose the **Resource Bundle** category to display the resource Bundle dialog, as shown in Figure 3–13.

Figure 3–13 Project Properties — Resource Bundle Dialog

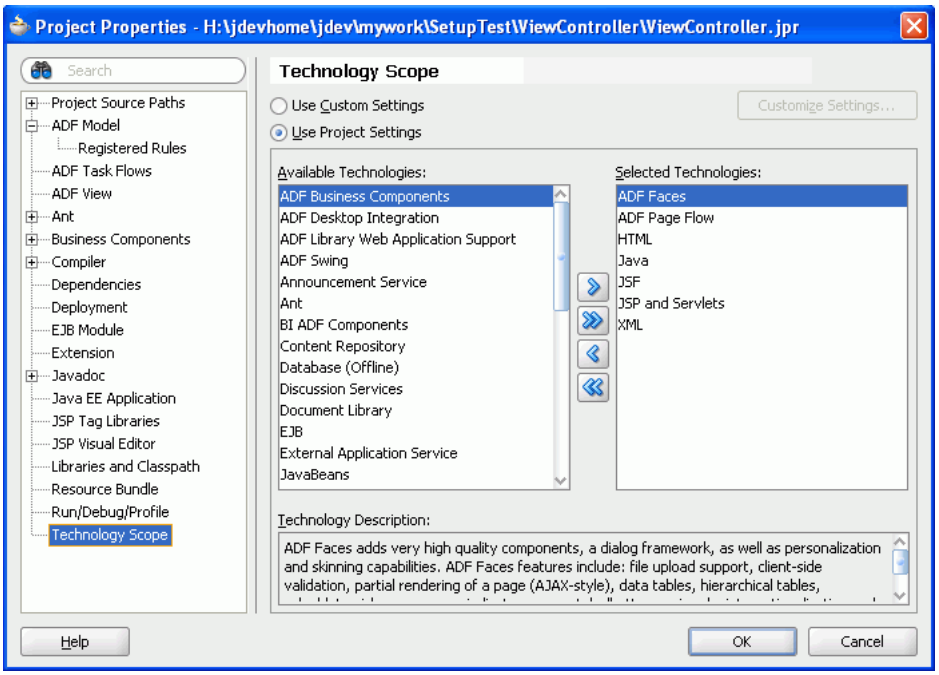


Verify that the **Resource Bundle Type** is set to *Xliff*.

> **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 3–14.

Figure 3–14 Project Properties — Technology Scope Dialog



Verify that the technology scopes that are selected in Figure 3–14 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 Section 3.9.1, "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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

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

- Create a Web Application Archive (WAR) deployment profile. To create the WAR deployment profile, see "How to Create Deployment Profiles" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
- Once you have defined the WAR, create an application archive (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 *Oracle Fusion Middleware Fusion Developer's Guide for 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. See "How to Install the ADF Runtime to the Application Server Installation" and "How to Create and Extend Oracle WebLogic Server Domains" in the *Oracle Fusion Middleware Administrator's Guide for Oracle Application Development Framework*.
- Deploy the application using one of these methods:

- Oracle Enterprise Manager Fusion Middleware Control
- WebLogic Scripting Tool (WLST) commands (see "Deployment Commands" in the *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*) or WebSphere Application Server (wsadmin) commands
- Command scripts and Ant scripts
- Oracle WebLogic Administration Console or IBM WebSphere Administrative Console

Part II

Defining Business Services

This part of the Developer's Guide discusses business services and service-oriented development, defaulting and derivation logic, creating validation rules, and using messages in Oracle Fusion Applications. It provides information about the Oracle Fusion Middleware extensions for Oracle Applications base classes and describes how to share reference data across organizations by using set IDs to partition the data into different sets of values. Also included is how to implement lookups and simple lookups.

The *Getting Started with Business Services* chapter provides overviews of ADF Business Components, services, validators, list of values (LOVs), and data types. It also discusses migrating PL/SQL to Java, batch processing, and extensibility and reusability.

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. The *Developing Services* chapter describes how to design the service interface, how to develop and invoke services. It also provides information about service versioning.

Defaulting logic means assigning attribute values when a row or entity object is first created or refreshed and is achieved either declaratively in the attribute's default field or programmatically by adding code to the `EOImpl` file. *Derivation* logic means assigning attribute values when some other attributes have changed. Derivation is achieved either declaratively in the transient attribute's default field or by using a validator, or programmatically by adding code to the `EOImpl` file. This chapter provides the information you need to determine whether to implement defaulting or derivation logic.

The *Message Dictionary* and *Messages Resource Bundles* are used to store messages for display from your application without hard-coding them into your forms and programs. By using the Message Dictionary and resource bundles you can define standard messages that you can use in all your applications, provide a consistent look and feel for messages within and across all your applications, define flexible messages that can include context-sensitive variable text, and change or translate the text of your messages without regenerating or recompiling your application code. The *Defining and Using Message Dictionary Messages* chapter provides an overview of Message Dictionary messages and discusses how to use them in Oracle Fusion Applications.

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 Middleware extensions support Oracle Applications features such as TL (translatable) table, WHO column, PL/SQL entity, FND services, Unique ID, and document sequencing. In JDeveloper,

selecting the Oracle Fusion Applications Developer role automatically sets the 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. This chapter describes the Oracle Fusion Middleware extensions for Oracle Applications base classes that extend the features of standard ADF Business Components classes.

Unique ID generation provides a mechanism to manage the key-generation process and to ensure that it runs without interruption. The process efficiently generates distinct sets of IDs in different databases for the same table, ensuring that the same key is never used for two different records created in different systems.

SetIDs enable different organizations within a single company 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. Each organization can maintain its job code data in the same table, using a set of values that is specific to that organization. You use set IDs to partition the table into different sets of values so that each organization can identify and access its own data. In addition to tables, other sources of reference data such as lookup types and views can also be partitioned and shared using set IDs. These are all generically referred to as reference entities. This chapter describes how to share reference data across organizations by using set IDs to partition the data, implement shared reference entities, extract and expose set ID metadata, and implement shared 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 Oracle applications.

A *document sequence* uniquely numbers documents generated by an Oracle Applications product. Using Oracle 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.

Implementing Audit Trail Reporting describes how to track the history of the changes that have been made to data in Oracle Fusion Applications. Audit Trail includes information such as who has accessed an item, what operation was performed on it, when it was performed, and how the value was changed.

This part contains the following chapters:

4

Getting 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:

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 Section 4.2, "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 *Oracle Fusion Middleware Fusion Developer's Guide for 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 addition 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.

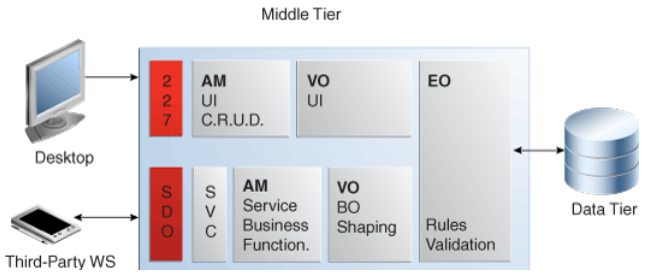
The methods on the application module are used to encapsulate task-level business logic, although these methods should delegate to the methods on entity objects whenever possible. If you have an option to put your business logic either on an entity object or an application module, then you should always 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.

Figure 4–1 UI Application Module and Service Application Module



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 Chapter 6, "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, the LOV view object should be added as a view accessor in the entity object, and the view accessor should be used to define entity level validators. The same view accessor is available at the view object level and should be used 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 *Oracle Fusion Middleware Fusion Developer's Guide for 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, then a combination of multiple techniques needs to be 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 should be used. 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 Chapter 5, "Developing Services."

4.7 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 utilize view criteria.

4.7.1 How to Define 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

4.7.1.1 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 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.

5 Developing Services

This chapter describes how customers can develop new business applications using existing services in the context of new or modified business processes.

This chapter contains the following sections:

5.1 Introduction to Services

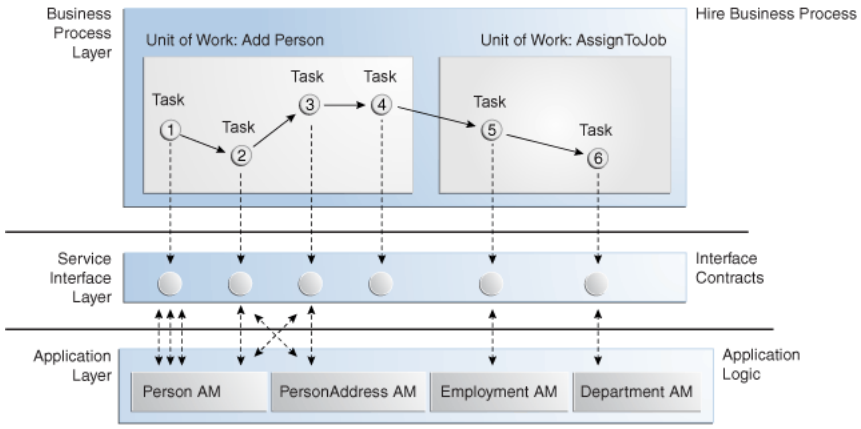
Service-oriented development is based on the concept of services. A service is defined in terms of its interface, which is the way the service is exposed to the outside world. This includes a set of parameters (defining data required for interaction with the service) and communication protocol used for data transfer and actual service invocation. The service interface is defined by a service name and a set of operations the service supports. Grouping of the methods in the service interface is defined by business functionality of the service.

The following characteristics are typical for services and should be reflected in the service interface:

- 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:

Figure 5–1 SOA Service — Business Service



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

Designing services from the service provider perspective include: how to identify business objects, service operations on business objects, and services, as well as how to define service exceptions and information.

5.2.1 How to Identify Business Objects

Identify which business objects that you want to expose from the service interface.

5.2.2 How to Identify Service Operations on the Business Objects

These are the actions that can be performed on business objects, such as *create* and *delete*. These operations are used to identify the service operations.

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.

  This group of operations are mainly used by the ADF Business Components Service framework for service based entity object and view object support. For more information about service-based entity objects and view objects, see Section 5.4.1, "How to Invoke a Synchronous Service."

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 Identifying 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. 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. If you only need to delete a child object, then you must have specific delete operations on the child objects. This is because you are not able to delete a child object using the delete method on its parent object. For example, `deletePurchaseOrder` will delete a purchase order and all of its lines, but won't only delete a specific line or lines.

5.2.2.3 Defining Service Operations - General Guidelines

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 Section 5.2.2.2, 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:

For `GetPersonName()` operation: The provider, at a minimum, requires the primary keys to access a person's name as input. However, the output could be either a formatted name in a form of a simple String or a complex document representing Name object. The list of returned attributes must be determined by the consumers' business requirements. If the first consumer of an interface only requires `FirstName` and `LastName` to be returned, it would be possible to only return these two values. However, it is likely that a popular service operation such as `GetPersonName()` will soon be adopted by more consumers, which will require other attributes of a name. In this example, it makes sense to include `Title`, `RoyalPrefix`, `LegalName`, and so on into the first specification of the service interface. This will avoid the creation of a new interface version in the foreseeable future.

Leverage standards wherever possible

Enterprise Business Object (EBO) standards, introduced by the AIA (Application Integration Architecture), are canonical forms of interfaces, which represent an abstract intermediary shape that integration parties go from and to in the integration. If existing Enterprise Business Object shapes are a good fit for the business requirements of your specific use case, they should be leveraged. Even if the EBO shape is not identical, the names of the EBO objects and attributes should be reused as much as possible. If your service needs to be consumed by either an internal stakeholder or an outside party, using a 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.

Service operation naming convention

Service operations should follow a consistent naming convention. A `verbNoun` syntax has proven ideal to express the behavioral aspect of a business object. Operation names should be meaningful and clearly express the function performed. They should not be generic or ambiguous. For example: A *Person* service operation name should not be `get()` or `invoke()`, but should be `getPersonName()`.

Service operation versioning

Changes to an interface can be either *compatible* or *incompatible*. If, for example, only optional attributes are added to an existing parameter, current consumers are not impacted. If however the parameter list changes, then this is an incompatible change.

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.

In order 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 once 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.

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:

Example 5–1 Service Operation on a List of Parameters

```
void updatePerson (Date BirthDate, PersonName[] Names, PersonAddress[] Addresses);
```

  – 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 needs to 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.

  Most custom methods may take business object primary keys as parameters. Complex business object documents should be passed primarily to the data-centric custom methods: `validatePersonName()`, `promoteEmployee()`, `formatPersonAddress()`, and so on. Primary keys should be passed only if you need the primary key information to look up the business object in the database, such as `terminateEmployee()` that takes an employee Id.

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 Section 5.2.2.2 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, once 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 How to Define Service Exceptions and Information

You can define service exceptions, partial failure and bulk processing, and informal messages.

5.2.4.1 Defining Service Exceptions

Once 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 Developing Services

After you design the service interface, you now must implement those services.

Note: This is from the service provider perspective.

5.3.1 How to Create Service Data Objects

In Oracle Fusion, service data objects (SDOs) are used to expose business objects in services. Each SDO must be backed by a view object. For more information about how to generate a service data object out of a view object, see the "How to Service-Enable Individual View Objects" section of the "Integrating Service-Enabled Application Modules" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

It is recommended that you have a separate view object for the service, rather than using the same view object in the UI. The dedicated service view object should represent the shape of your business object.

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.

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 Section 5.2.4.2, "Defining Partial Failure and Bulk Processing."

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.

5.3.2 How to Create Services

The service interface is generated from an ADF application module. For more information, see "Integrating Service-Enabled Application Modules" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Oracle recommends that you have a separate application module for the service, which is different from the application module used in the UI.

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 5–1 Return Objects Examples

Operation without Informational Messages (*Support Warnings* Flag is off)	Operation with Informational Messages (*Support Warnings* Flag is on)	Comments
`List<Person> processPerson(String op, List<Person> persons, ProcessControl ctrl)`	`PersonResult processPerson(String op, List<Person> persons, ProcessControl ctrl)`	`PersonResult` contains a list of `Persons`, and a list of `ServiceMessages`.
`Person createPerson(Person person)`	`PersonResult createPerson(Person person)`	The list of `Person` in `PersonResult` should contain only one element.
`void terminateEmployee(BigDecimal empId)`	`ServiceMessage terminateEmployee(BigDecimal empId)`	
`String getApplicationName(BigDecimal applicationId)`	`StringResult getApplicationName(BigDecimal applicationId)`	The `StringResult` contains a `String` and a list of `ServiceMessages`.

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

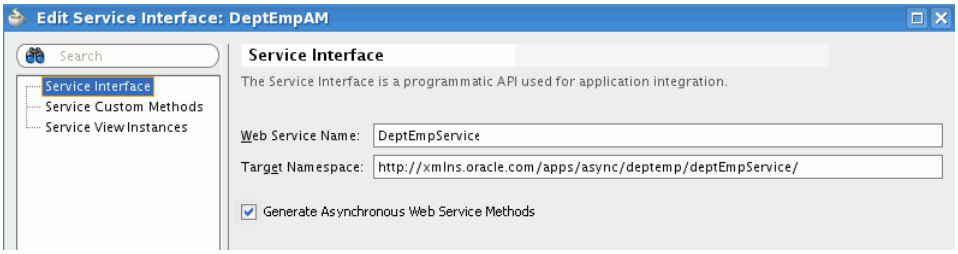
5.3.3 How to Generate Synchronous and Asynchronous Service Methods

Each service method can be exposed as both synchronous version and asynchronous version.

To generate synchronous and asynchronous service methods:

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.

5.4 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), and the payload is still small.
- 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 How to Invoke 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.

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 (ESS), then using a service factory is recommended. It is easier to write a service client using service factory than using JAX-WS. If the service is co-located, the service invocation is more performant because it does not invoke XML serialization and de-serialization.

For more information about invoking a service using service factory, see Chapter 41, "Synchronously Invoking an ADF Business Components Service from an Oracle ADF Application."

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 Chapter 39, "Working with Data from a Remote ADF Business Components Service."

Using 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.

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 Part VI, "Common Service Use Cases and Design Patterns".

Caution: For more information, see the "*How to Create Service-Enabled Entity Objects and View Objects*" section in the Integrating Service-Enabled Application Modules chapter of the *Oracle Fusion Middleware Fusion Developer's Guide for 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 How to Invoke an Asynchronous Service

If you need to invoke an asynchronous service, then you must use BPEL. Invoking an asynchronous service from Java is not allowed. 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 Chapter 42, "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.

6

Defining 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:

- Section 6.1, "Understanding Entity Object Defaulting and Derivation Logic"
- Section 6.2, "Using Groovy Scripting Language"
- Section 6.3, "Using Oracle ADF Validators and Convertor Hints"

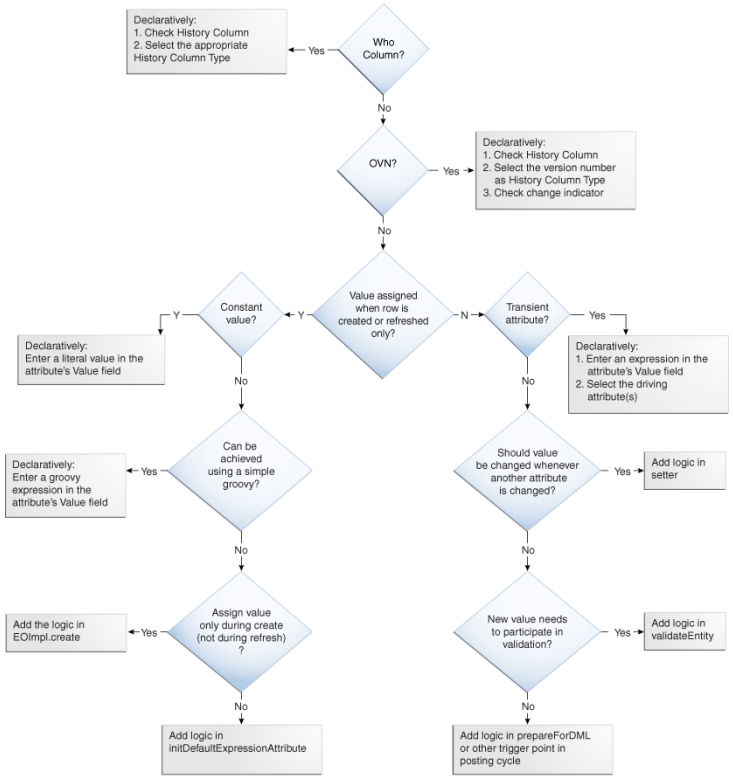
6.1 Understanding Entity Object Defaulting and Derivation Logic

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.

Figure 6–1 Defaulting and Derivation — Decision Tree



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:

```
Row parentRow = ....RowIterator ri =
(RowIterator)parentRow.getAttribute("<childEOAccessorName>");
 Row childRow = r1.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 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. (No UI support at this time).
- **Attribute Default Values** - Define a default value expression on an attribute using Groovy script. (No UI support at this time).
- **Variables** - Define a variable on an entity whose value is computed using Groovy script. (No UI support at this time).

6.2.1 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 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

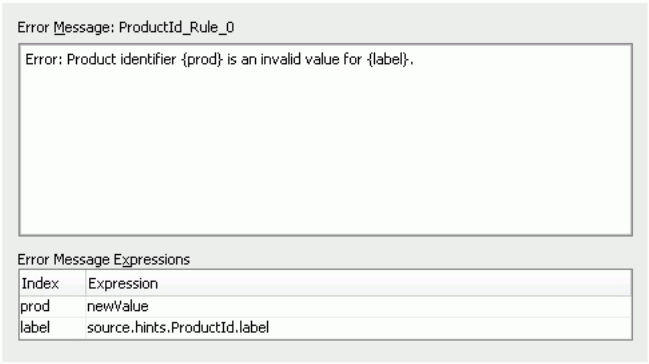
Instead of using the following SQL to achieve this:

```
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:



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 6–1 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 is an example of how to average a collection.

Example 6–2 Averaging a Collection

```
attribute Number EmpSal : SAL
{
  expressionValidator(expression =
    "newValue <= source.createUnqualifiedRowSet().avg(\"EmpSal\") * 1.2");
}
```

Example 6–3 is an example of a built-in or custom method call on the `sourceObject` of this validator (`sourcObject` being the Entity on which this validator is being run). `isAttributeChanged(String)` is a public method on the `EntityImpl`:

Example 6–3 Built-in or Custom method Call

```
if (source.isAttributeChanged("EmpSal") || source.isAttributeChanged("EmpComm"))
           {
              return true;
           }
           return false;
```

Example 6–4 is an example of getting to oldValue / newValue of an attribute on which this validator is applied:

Example 6–4 Getting to Old Value and New Value of an Attribute

```
return (oldValue == null || newValue < olValue * 1.2);
```

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 6–5 Getting the Old Value of a Transient Entity Object Attribute

```
index = object.getStructureDef().getAttributeIndexOf("Salary");
```

```
return object.getAttribute(index, oracle.jbo.server.EntityImpl.ORIGINAL_VERSION);
```

Example 6–6 is an example of the `WHILE` construct as well as calling an accessor (Emp):

Example 6–6 While Construct and Calling an Accessor

```
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 6–7 Simple Transient Attribute

```
attribute transient Integer YearSal
{
  transientexpression = "EmpSal * 12";
}
```

Example 6–8 Sum or Count a Collection

```
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

```
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 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 6–10 Attribute Value Defaulting

```
<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)
      {
        retune 500;
      }

]]></TransientExpression>
```

Example 6–11 Attribute Defaulting with a Transient Attribute Calculation Expression

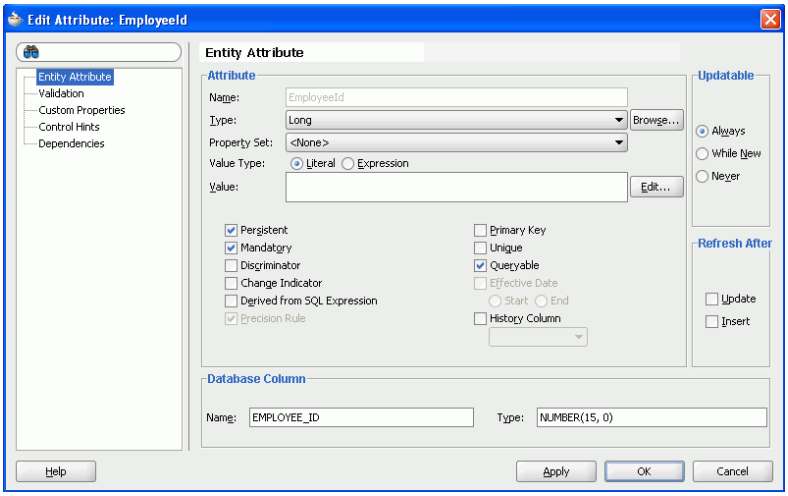
```
<ViewAttribute
```

```
    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>
```

6.2.4 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".

Figure 6–3 Entity Object — Expression Editor

To access the Expression Editor, click the **Edit** button located next to the **Value** text box.

Note: Recalculate Expression is used to determine whether or not the expression needs to 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

In some situations, you should consider using Oracle ADF validators or converter hints instead of using messages.

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.

How to 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 *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

7

Defining and Using 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:

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, pages, or business objects, and indicate that processes and actions are performing or are completed.

All other messages can be stored in resource bundles, with the exception of strings and messages that need to be accessed by C or PL/SQL programs (resource bundles only can be accessed by Java programs). These exceptions can be stored in the Message Dictionary as *Informational* 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 Section 7.2, "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 Section 7.3.4, "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 to how to generate incidents and log entries from Message Dictionary messages, see Section 7.5, "Understanding Incidents and Diagnostic Logs with Message Dictionary." For information about how incidents and log entries can be used, see the "Managing Oracle Fusion Applications Log Files and Diagnostic Tests" chapter and the "Introduction to Troubleshooting Using Incidents, Logs, QuickTrace, and Diagnostic Tests" chapter in the *Oracle Fusion Applications Administrator's Guide*.

You use the Manage Messages task in the application's Setup and Maintenance work area to create and maintain Message Dictionary messages. For more information, see the *Oracle Fusion Applications Common Implementation Guide*.

Oracle Fusion Applications provide some common messages in the Message Dictionary with message names that begin with `FND_CMN_`. You should ensure that you 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 Section 7.3.4, "About Message Components." For information about logging and incident creation, see Section 7.5, "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 *Oracle Fusion Applications 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

```
+---------------------------------------------------------------------------------------
Shipping Execution: Version 15.0 - Development

Copyright (c) 1979, 2008, Oracle Corporation. All rights reserved.

WSHRDPIK module: Pick Slip Report
+---------------------------------------------------------------------------------------

Current system time is 10-DEC-2008: 14:50:10

+-------------------------------------------------

Process monitor session started : 10-DEC-2008 14:40:48
Process monitor session ended : 10-DEC-2008 14:45:30

The following errors were encountered
+----------------------------------------

The routine AFPCAL received a failure code while parsing or running your
concurrent process. Make sure that the arguments are passed in the correct format.
```

- Strings for information level diagnostic logging.

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.

You either can use the Information type for all non-error and non-warning messages, or you can choose to store complete messages as Information messages and fragments as UI String messages. 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.

7.3 Understanding Message Content

Messages must have unique *message names*. Although *message numbers* are not required, you should use them for error messages in order 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 not required, but can be used to store notes about context and use of the message. Translation notes can also be used to provide information to help translators understand how the message is used and thus provide a more accurate translation.

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 run time. They are used as a way 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 Section 7.5, "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 occurred as a result, 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 Section 7.3.5, "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.

Although this component is optional, it should be used for errors that require help desk processing, and should 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 in order 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 in order 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 brackets 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".

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 product functionality, setup and maintenance. Such messages are typically routed to functional administrators or product super users.
  - *System* - This value refers to the system, database, technology stack, and so on. Such messages are typically routed to 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 routed to 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.
  - *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 more information about how customers use incidents, see the "Managing Oracle Fusion Applications Log Files and Diagnostic Tests" chapter and the "Introduction to Troubleshooting Using Incidents, Logs, QuickTrace, and Diagnostic Tests" appendix in the *Oracle Fusion Applications Administrator's Guide*.

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 is enabled.

You 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, you use methods in the `fddutl` package.

For more information about the Message Dictionary APIs, see the "Message Dictionary" chapter in the *Oracle E-Business Suite Developer's Guide*. You can download this soft-copy documentation as a PDF file from the Oracle Technology Network at `http://www.oracle.com/technetwork/indexes/documentation/`

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.

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, 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 *Oracle Fusion Middleware Administrator's Guide*.

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 How to Raise 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

Exception Class	Message Dictionary Class
JboException	oracle.apps.fnd.applcore.messages.ApplcoreException
RowValException	oracle.apps.fnd.applcore.messages.ApplcoreRowValException
AttrValException	oracle.apps.fnd.applcore.messages.ApplcoreAttrValException

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

Token Type	Token Value Object Type
TEXT	java.lang.String
NUMBER	java.math.BigDecimal
DATE	java.sql.Timestamp

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 Section 7.9, "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`.

Example 7–1 ApplcoreException

```
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

```
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

```
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);
tokens.put("DATE_TOKEN", new Timestamp(cal.getTimeInMillis()));

ApplcoreAttrValException ex = new ApplcoreAttrValException("MYAPP:::MY_MESSAGE_
NAME",
                                "MyEODefName","AttrName", "AttrValue", tokens);
```

Example 7–4 Native JBOException

```
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;
```

7.6.2 How to Retrieve 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 7–5.

Example 7–5 Retrieving Messages

```
// 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);

// Get the token substituted full admin message, in plain text format.
String adminText = Message.getAdminText("MYAPP", "MY_MESSAGE", tokens);

// Get the token substituted full admin message, in HTML format.
String adminHTML = Message.getAdminHTML("MYAPP", "MY_MESSAGE", tokens);
```

7.7 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 once 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 *Oracle Fusion Middleware Fusion Developer's Guide for 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 Section 6.2, "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:

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 the **Select Message** button 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 only search messages 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.

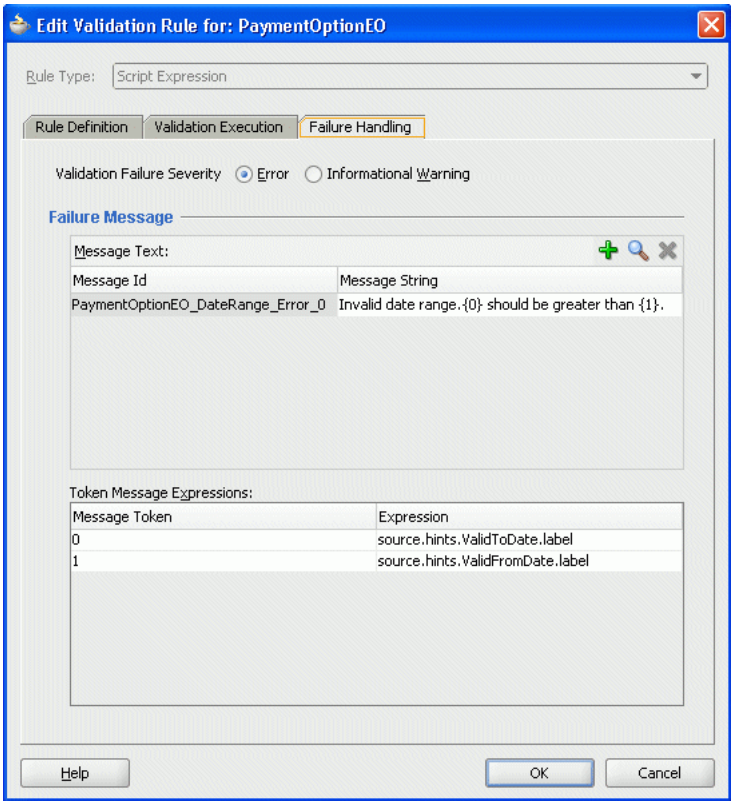
   > **Note:** If the selected message contains tokens, a row for each token is added into the **Error Message Expressions** table.

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 Section 6.2, "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



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 Programmatically" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

7.8 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 How to Raise 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 How to Raise Errors in PL/SQL

The FND_MESSAGE PL/SQL package allows you to set one message and its tokens, as shown in Example 7–6. It also allows you to set multiple messages in the stack by explicitly pushing the current message onto the stack, as shown in Example 7–7. When you need to retrieve the message from the stack, an explicit pop() is required, as shown in Example 7–8.

Example 7–6 Getting Message and its Tokens

```
-- 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 7–7 Receiving a Message Record and Clearing Message

```
-- setting INVALID_USER as the current message
fnd_message.set_name('FND', 'INVALID_USER');
-- setting token value for NAME
fnd_message.set_token('NAME', 'TESTUSER');
.......
-- 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 subsituted 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 7–8 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 How to Retrieve 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 7–9.

Example 7–9 Calls to OAExceptionUtils.CheckError() — Unknown Error Handling Style

```
import oracle.apps.fnd.applcore.common.OAExceptionUtils;
  ...

  try
  {
    // Create and execute a plsql statement
    String mystmt = "BEGIN MY_PLSQL_PACKAGE.MY_PROCEDURE(); END;";

    DBTransaction txn = getDBTransaction();
    CallableStatement mystmt = txn.createCallableStatement(mystmt, 1);

    myStmt.executeUpdate();

    // Check for errors left on message stack without raising exception
    OAExceptionUtils.checkErrors(txn);
  }
  catch(SQLException sqlE)
  {
    // Check for FND Messages exception.
    // FND Messages exception always has error code -20001.
    if (sqlE.getErrorCode() == 20001)
    {
      OAExceptionUtils.checkErrors(txn);
    }
    else
      // Not a FND Messages exception, re-raise.
      throw sqlE;
  }
```

7.9 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 occured. 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

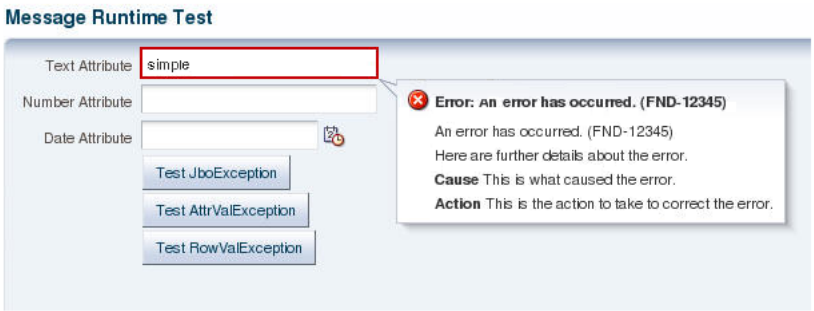
For information about finding the cause of an error and its corrective action and for information about viewing and managing log files, see the "Managing Log Files and Diagnostic Data" chapter and the "Introduction to Troubleshooting Using Incidents, Logs, QuickTrace, and Diagnostic Tests" appendix in the *Oracle Fusion Middleware Administrator's Guide*.

7.10 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 How to Programmatically Convert 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.applcore.messages.model.util.Util`.

Sample code is shown in Example 7–10.

Example 7–10 Converting XML Messages to HTML or Plain Text

```
Exception ex =
      new ApplcoreException("FND:::MY_TEST_MESSAGE");

    // Retrieve the HTML short message
    String htmlShort = Util.formatHTMLMessage(ex);

    // Retrieve the HTML message details.
    String htmlDetails = (Util.formatHTMLDetailMessage(ex)).getHTMLText();

    // Retrieve the plain text messge details
    String textDetails = (Util.formatHTMLDetailMessage(ex)).getText();

     // Retrieve the full plain text message
    String textMsg = Util.formatTextMessage(ex);
```

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 by configuring the error format handler:

1. Under the user interface project, open the `DataBindings.cpx` file.

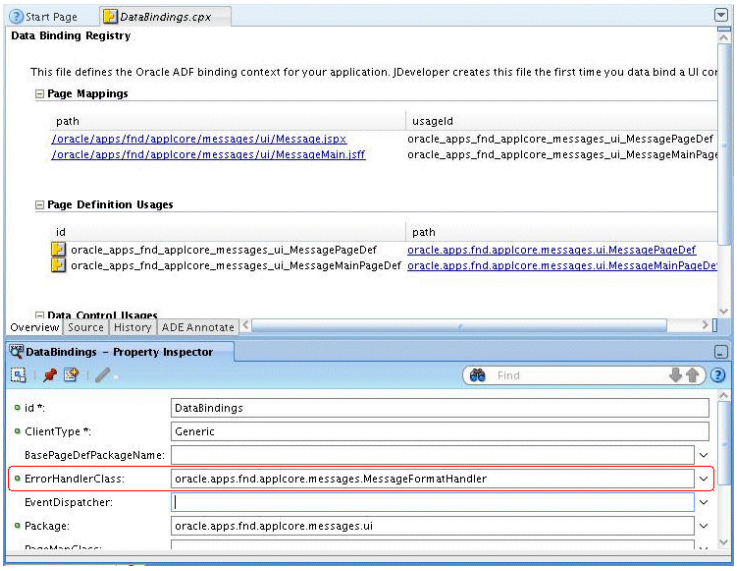
   Tip: JDeveloper names the user interface project `ViewController` by default.

2. In the Property Inspector, set the **ErrorHandlerClass** field to the value shown in Example 7–11 and Figure 7–4.

Example 7–11 The Value of the `ErrorHandlerClass` Field

```
oracle.apps.fnd.applcore.messages.MessageFormatHandler
```

Figure 7–4 Setting the Value of the ErrorHandlerClass Field in the Property Inspector



7.11 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 more information, see the *Oracle Fusion Applications Common Implementation Guide*.

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 Chapter 49, "Implementing Function Security."

Table 7–3 lists the task flows and their parameters.

Table 7–3 Messages Task Flows and Parameters

Task Flow Name	Task Flow XML	Parameters Passed	Behavior	Comments
Manage Messages	/WEB-INF/oracle/apps/fnd/applcore/messages/ui/flow/ManageMessagesTF.xml#ManageMessagesTF	mode='search' [moduleType] [moduleKey] mode='edit' messageName applicationId [pageTitle]	Search mode launches the search page, with optional parameters to restrict to a particular module. Edit mode launches the edit page for a particular message. The `messageName` and `applicationId` parameters are mandatory as they specify the message to edit.	NA.

8

Managing Reference Data with SetIDs

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:

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 should be used 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 Chapter 10, "Implementing Lookups".

SetID Implementation

Once 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 Section 8.3, "Integrating SetID Task Flows into Oracle Fusion Functional Setup Manager".

For information about how to use the setID setup applications, see the *Oracle Fusion Applications Common Implementation Guide*

8.1.1 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 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 Section 8.2.3, "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 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:

  ```
  WHERE SET_ID = :1
  ```

- **Row striping with common rows** (COMMON) — This is exactly the same as the row striping pattern, with the addition of a COMMON partition. You filter as follows:

  ```
  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 needs to 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:

  ```
  SET_ID: WHERE PARENT.PK1=SUBSCRIPTION.PK1 AND SUBSCRIPTION.SET_ID = :1
  ```

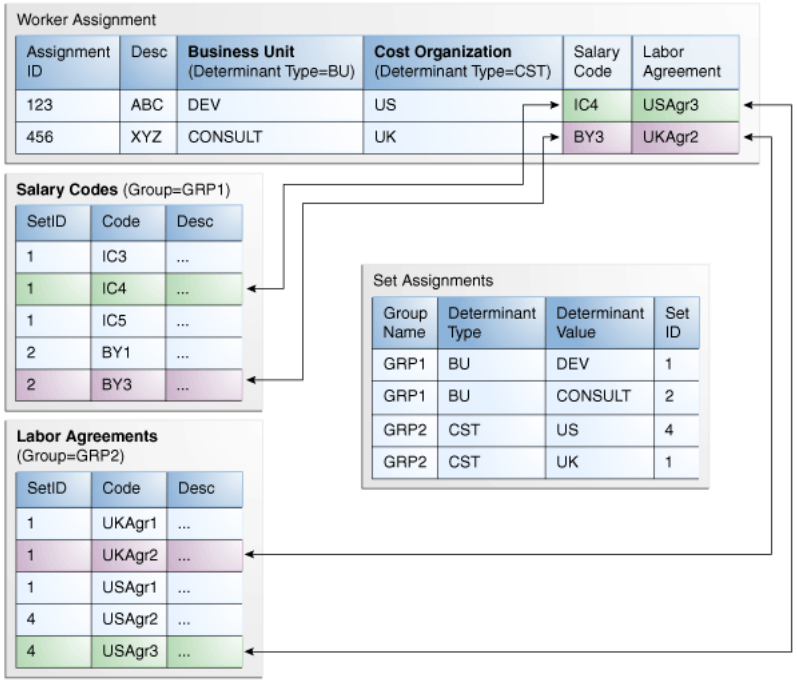
For more information about partitioning patterns, see Section 8.2.1, "How to Annotate Reference Entity Objects for Sharing".

8.1.3.2 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.

Figure 8–1 Example of a Table with Two SetID Reference Groups



8.1.3.3 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 Section 8.1.3.2, "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 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**

```
/**
 * 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**

```
/**
 * 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**

```
/**
 * 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
 * @return the corresponding reference group name
 */
function getReferenceGroupName(X_REFERENCE_ENTITY_NAME in varchar2) return
varchar2;
```

- **isValidSet**

```
/**
 * Returns true if the set ID exists in the FND_SETID_SETS table,
 * false if not
 *
 * @param setId The setId value
 * @return Boolean
 */
function isValidSet(X_SET_ID in number) return Boolean;
```

8.2 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 Section 8.1.3.1, "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 **oracle.apps.fnd.applcore.oaext.model**.

  For more information, see the "Creating a Business Domain Layer Using Entity Objects" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for 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 Section 8.1.3.3, "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 Section 8.1.3, "Understanding SetID Machinery".

8.2.1 How to Annotate Reference Entity Objects for Sharing

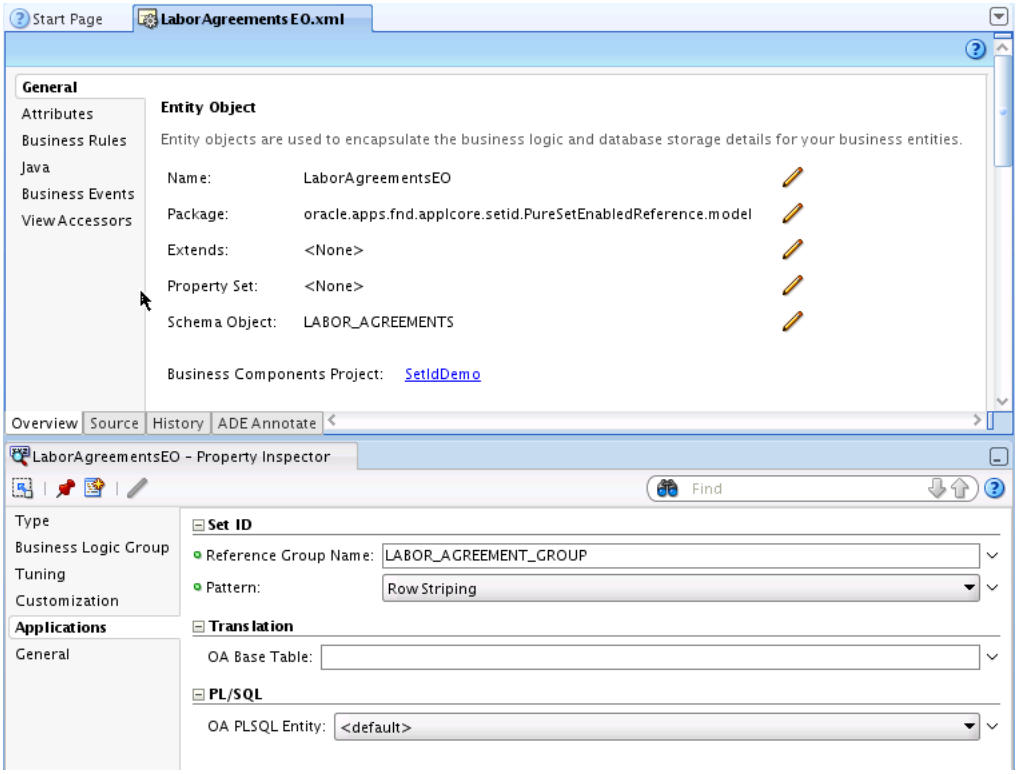
Note: These annotations are required only for *set-enabled* non-lookup reference entities. Lookup references and set-enabled lookup references use a predefined lookups pattern; the reference group name is retrieved directly from database by the lookup code and the view application ID that are set on 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:

1. Double-click the entity object to access its properties, as shown in Figure 8–2.

Figure 8–2 Entity Object SetID Properties



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 Section 8.1.3.2, "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`

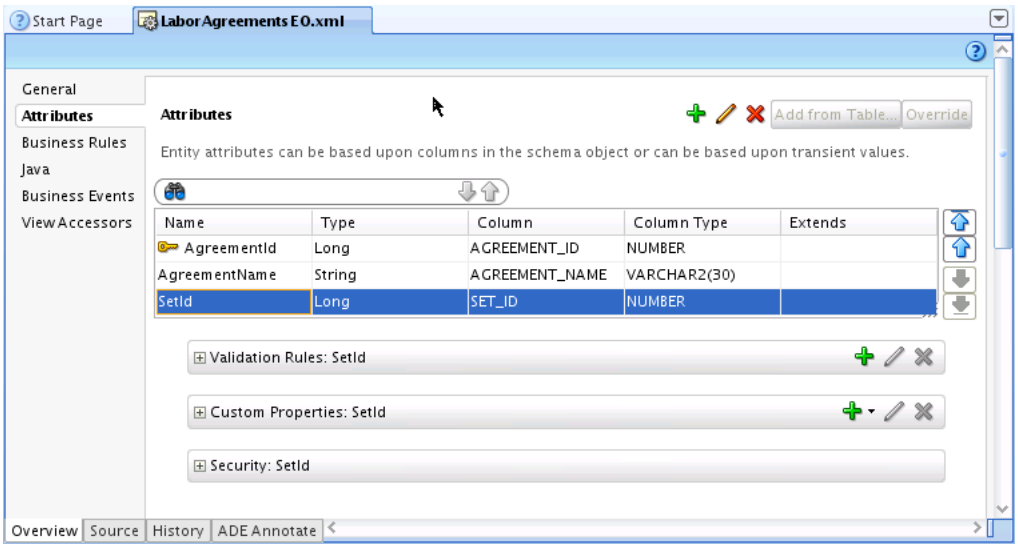
     When you select the `SUBSCRIPTION` pattern, the **SetID Reference Table Pattern** field appears. Specify the subscription table to use.

     Important: The primary key columns of the setID subscription table must be named exactly the same as those in the reference entity, and the setID column must be named `SET_ID`.

   For more information about these options, see Section 8.1.3.1, "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.

Figure 8–3 Entity Object SET_ID Attribute



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 How to Build 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.

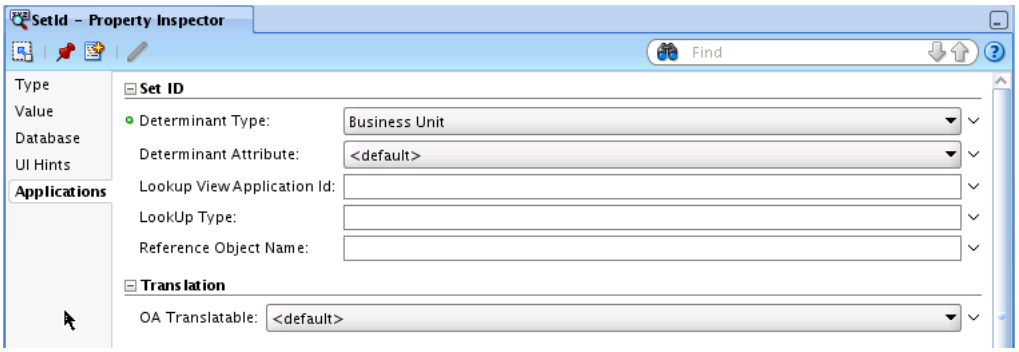
8.2.3 How to Annotate Transactional Entity Objects for SetID

After you create entity associations for foreign references, you annotate the transactional entity object.

To annotate the transactional entity object:

1. Double-click the entity object to access its properties, as shown in Figure 8–4.

Figure 8–4 Business Unit as the SetID Determinant Type



2. In the Applications section of the Property Inspector, designate which attributes are setID determinants for the table. For more information, see Section 8.1.2, "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

Code	Determinant Type
AB	Asset Book
BU	Business Unit
CST	Cost Organization
PU	Project Unit
SET	Reference Data Set

Notes:

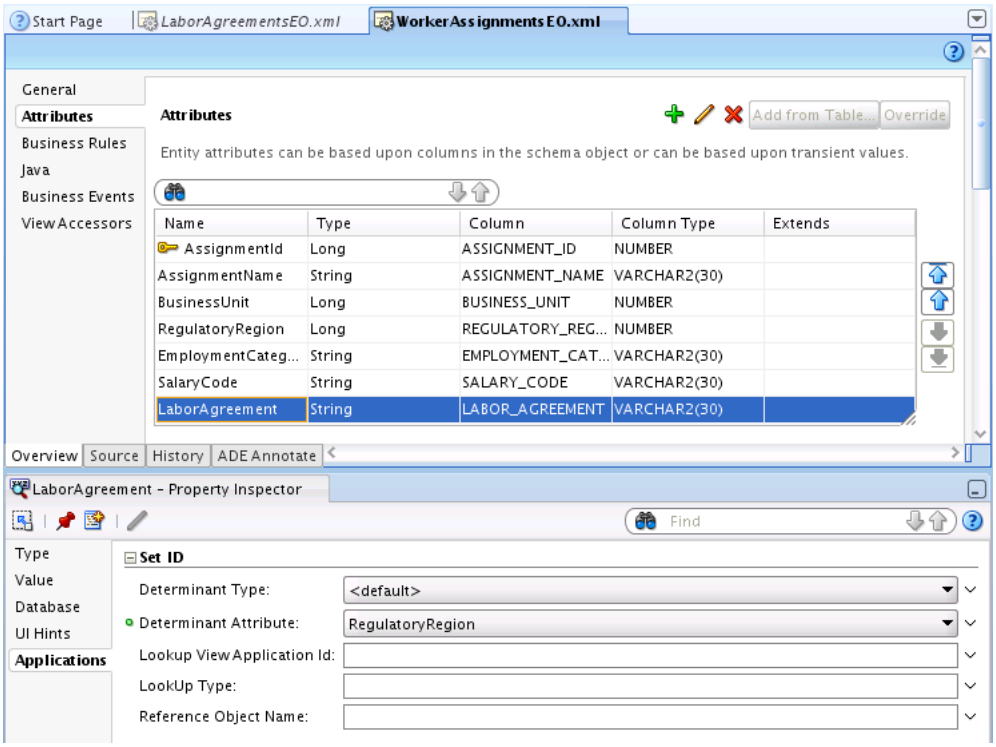
- The attribute you need to use as a determinant might not exist on the parent record where its value can be retrieved from some context. In that case, you must create a transient attribute to represent the determinant, set the **SetId Determinant Type** property for it, then override the getter method of the transient attribute to get the value from wherever it has been stored.
- Once 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



Attention: 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 How to Define 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

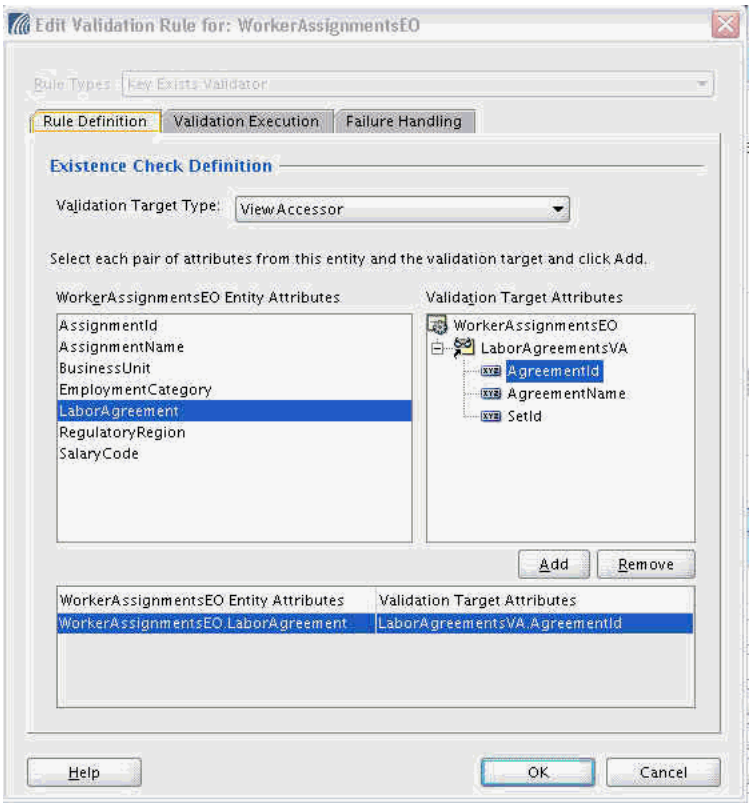
The key exists validator will include the mapping of the foreign key attributes in the transactional entity to the corresponding attributes in the reference view accessor.

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 key exists validator:

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`.

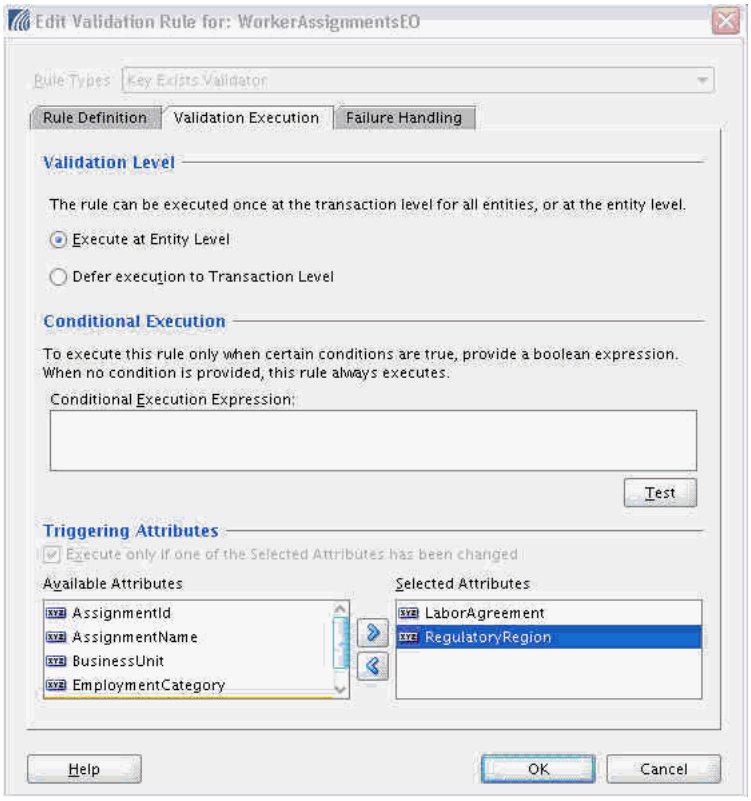
Figure 8–6 Foreign Reference Validation Rule Definition



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



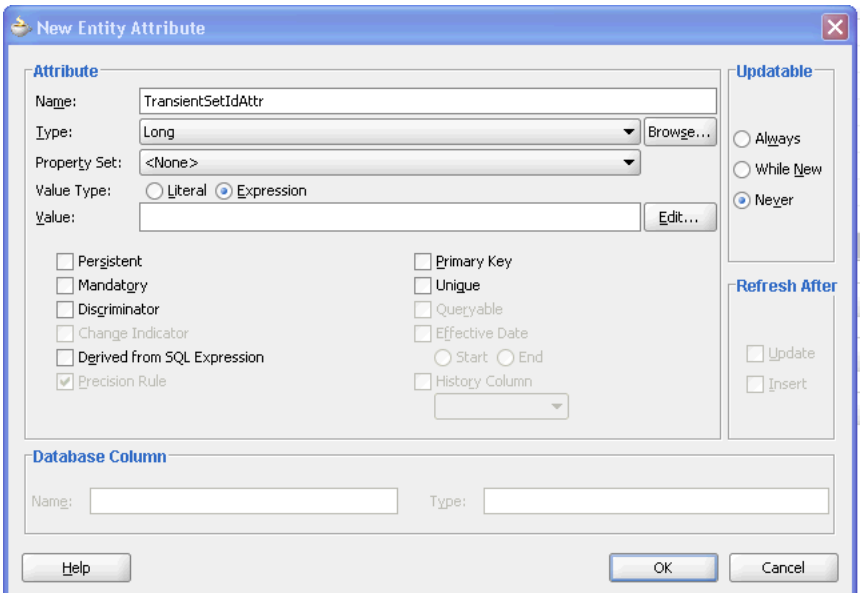
Note: The foreign key attributes that were mapped on the Rule Definition tab are by default added as triggering attributes.

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.

To create a transient setID attribute:

1. Create a new transient setID attribute to map to the SetId attribute on the reference entity, as shown in Figure 8–8.

Figure 8–8 New Transient SetID Entity Attribute



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:

```
/** 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.

8.2.6 How to Create LOVs for Shared Reference Entities

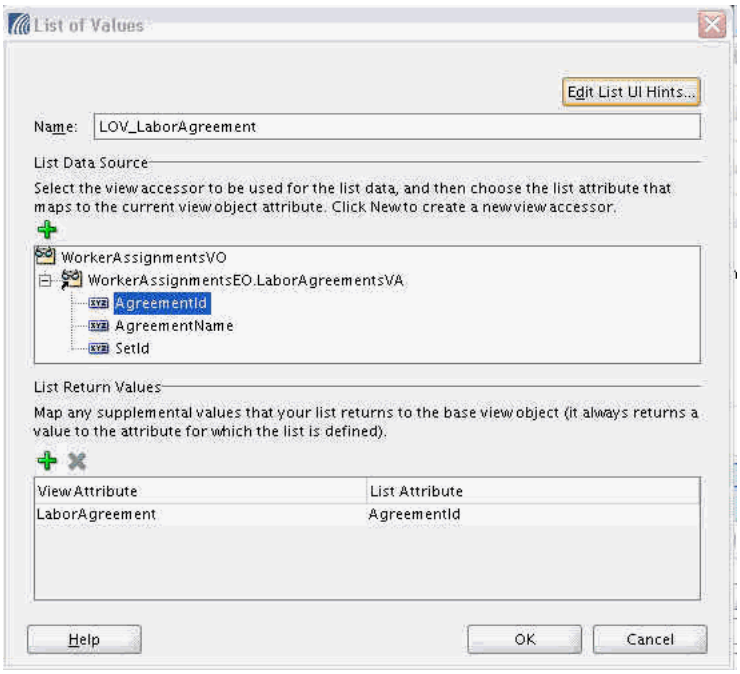
ADF Business Components supports defining LOVs at the attribute level in view objects.

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 `LOV_attribute_name`, as shown in Figure 8–9.

Figure 8–9 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 **Edit List UI Hints** to access the List UI Hints dialog.

8.3 Integrating SetID Task Flows into Oracle Fusion Functional Setup Manager

Every 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 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. 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

Task Flow Name	Task Flow XML	Parameters Passed	Behavior
Manage Reference Data Sets	`/WEB-INF/oracle/apps/fnd/applcore/setid/publicUi/flow/ManageSetIdSetsTF.xml#ManageSetIdSetsTF`	To optionally specify a page heading for the task flow: `pageTitle='titlestring'`	This task flow enables you to create and update reference data sets (setIDs and codes).

Manage Reference Data Set Assignments	`/WEB-INF/oracle/apps/fnd/applcore/setid/publicUi/flow/ManageSetIdAssignmentsTF.xml#ManageSetIdAssignmentsTF`	To invoke the task flow:
		`determinantType=type`
		
		To optionally restrict the page to assignments for a single reference group:
		`referenceGroupName=name`
		
		To optionally specify a page heading for the task flow:
		`pageTitle='titlestring'`
			This task flow enables you to manage reference data set assignments for a particular determinant type.

For more information about task flows, see the *Oracle Fusion Applications Common Implementation Guide*.

9

Using Fusion Middleware Extensions for Oracle Applications Base Classes

This chapter describes the Fusion Middleware extensions for Oracle Applications base classes that extend the features of standard ADF Business Components classes.

The chapter includes the following sections:

9.1 Introduction to Fusion Middleware Extensions for Oracle Applications Base Classes

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 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 Fusion Middleware extensions are the following:

- `OAApplicationModuleImpl`
- `OAEntityImpl`
- `OAEntityDefImpl`
- `OAViewObjectImpl`
- `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 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 Chapter 2, "Setting Up Your Development Environment."

9.2 Using 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 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 `AViewRowImpl` 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:

- Task 1, "Create an entity object for a _TL table"
- Task 2, "Create an entity object for a base table"
- Task 3, "Associate the _VL view and _TL table entity objects"
- Task 4, "Create a view object that uses a translatable entity"

Task 1 Create 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 Task 2, "Create 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.

Task 2 Create 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.

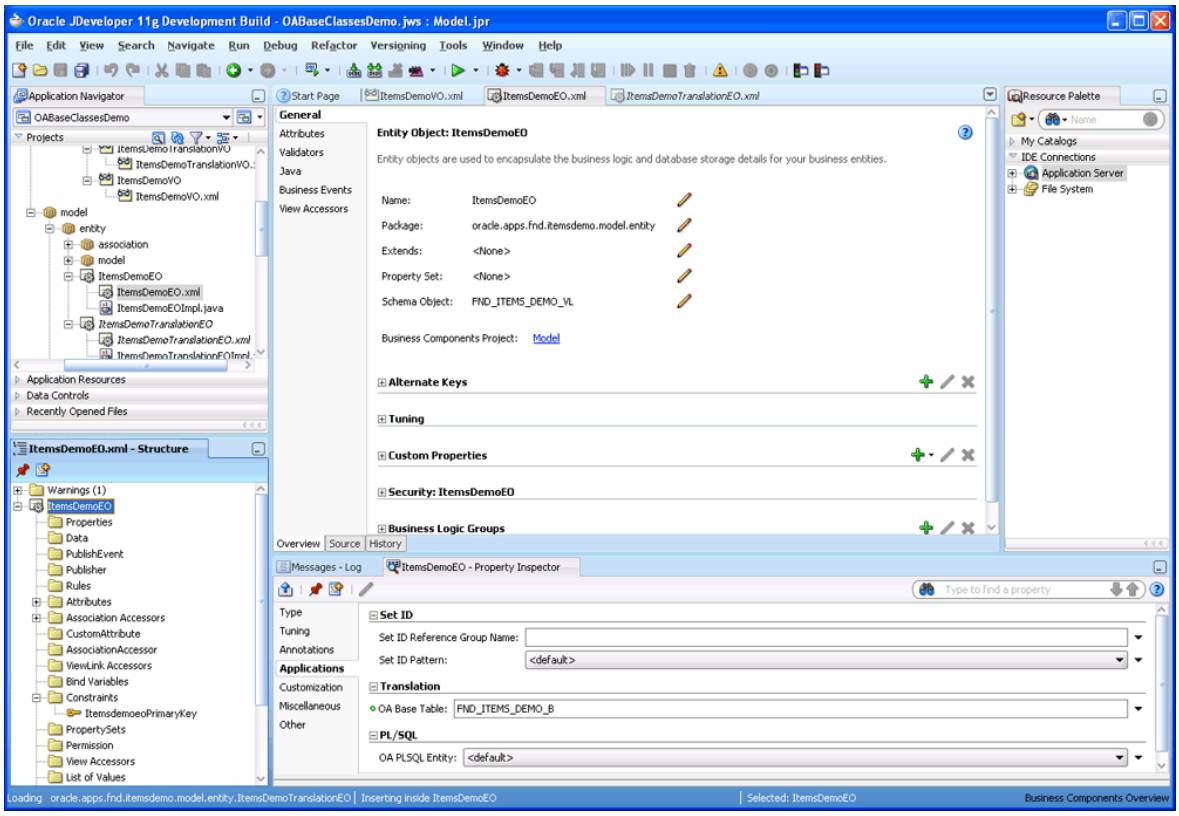
3. Identify your primary keys as you normally would.

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



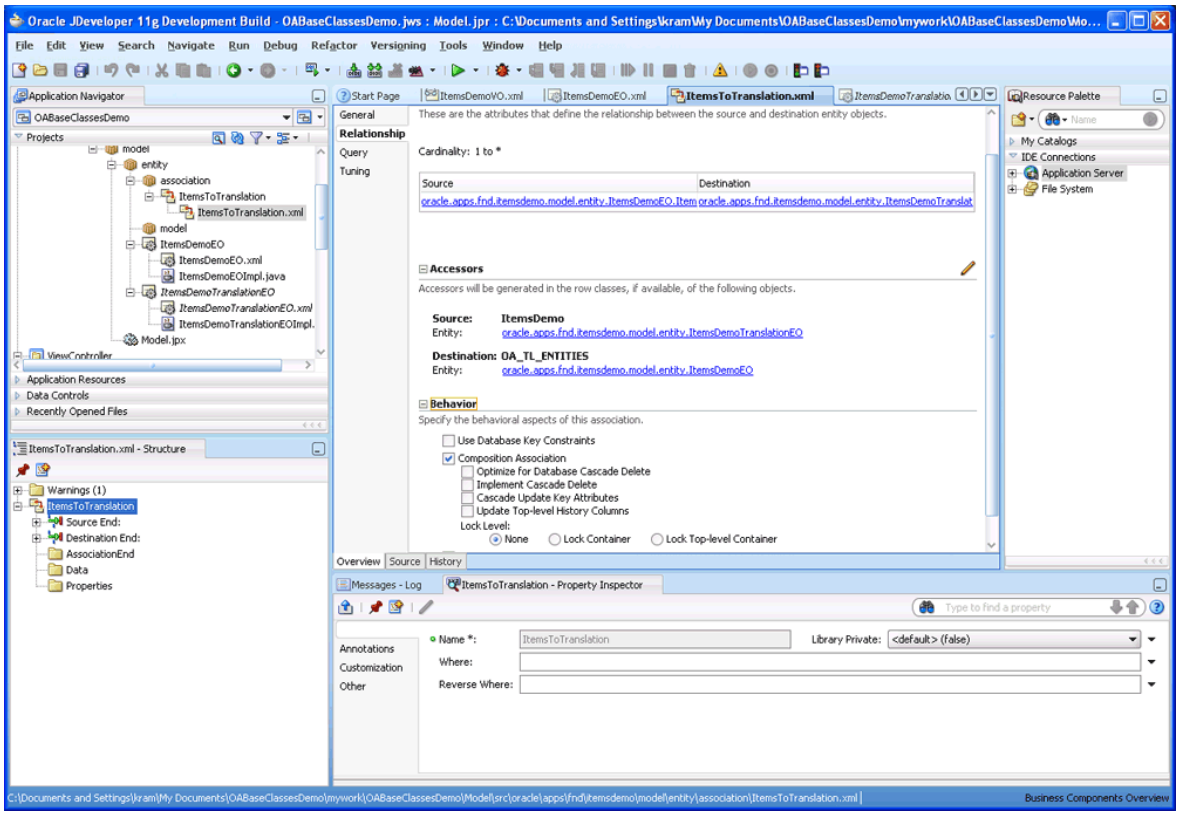
Oracle Fusion Middleware Extensions for Applications automatically overrides the entity's `doDML()` method to ensure that all inserts, updates, and deletes are actually performed on the base table identified by this property. All reads will be done against the _VL view.

Task 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.

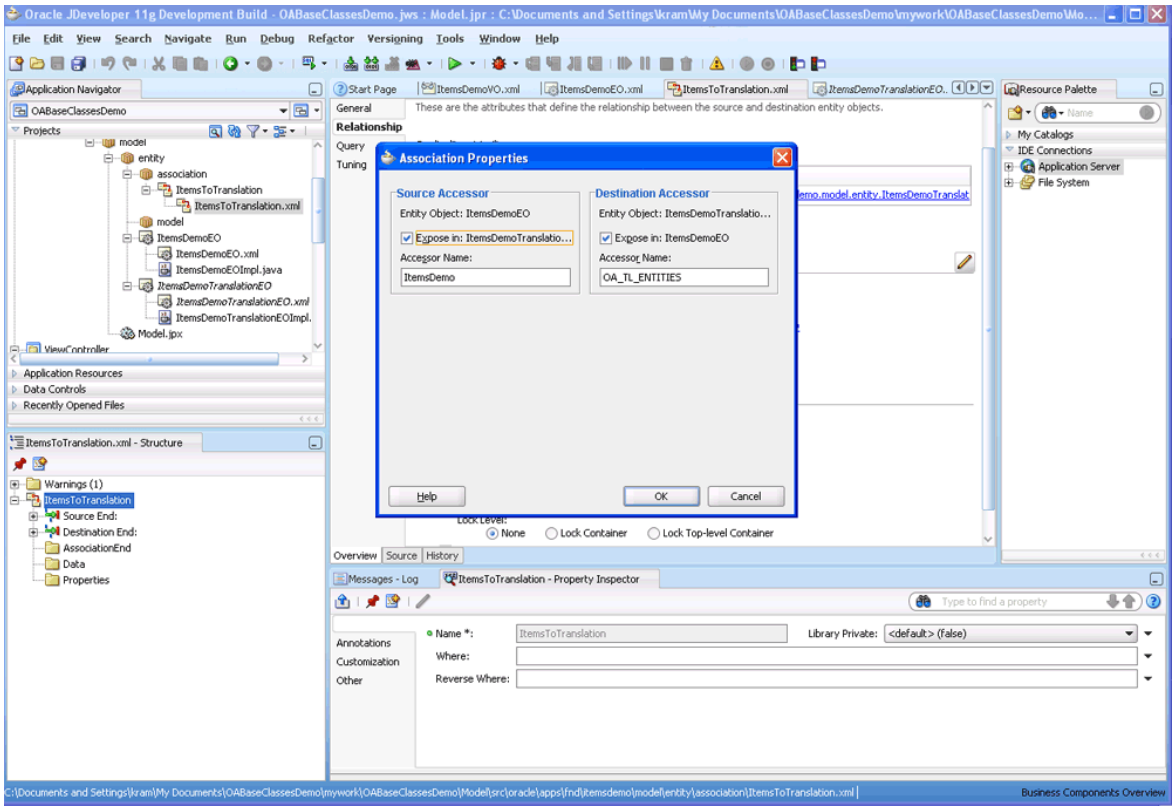
5. Select the base entity as the source and the _TL entity as the destination.

   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.

Figure 9–4 Association Properties



Task 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:

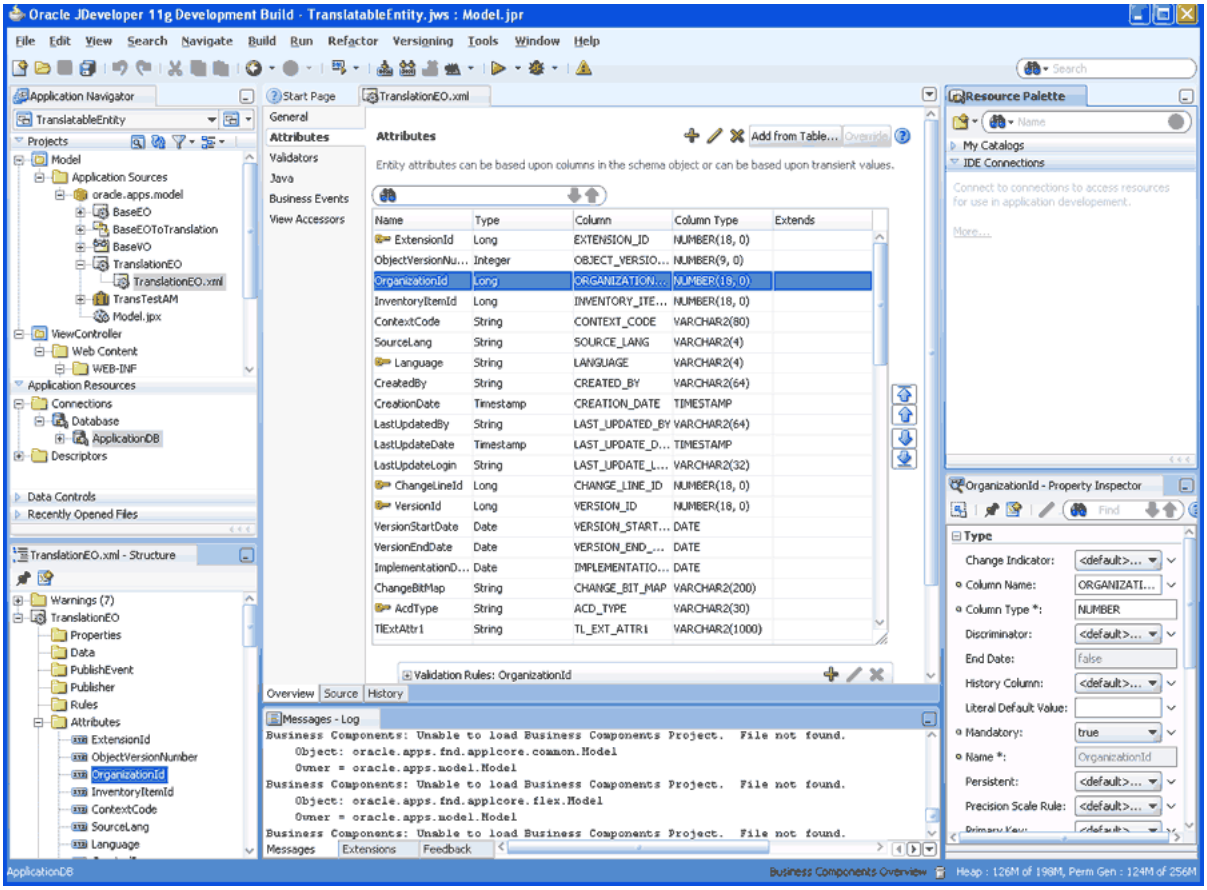
There may be a rare case where you have a _TL table and _VL view and no _B table, because all of the attributes are translatable. If this occurs, do the following:

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.

9.2.2.1 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 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.

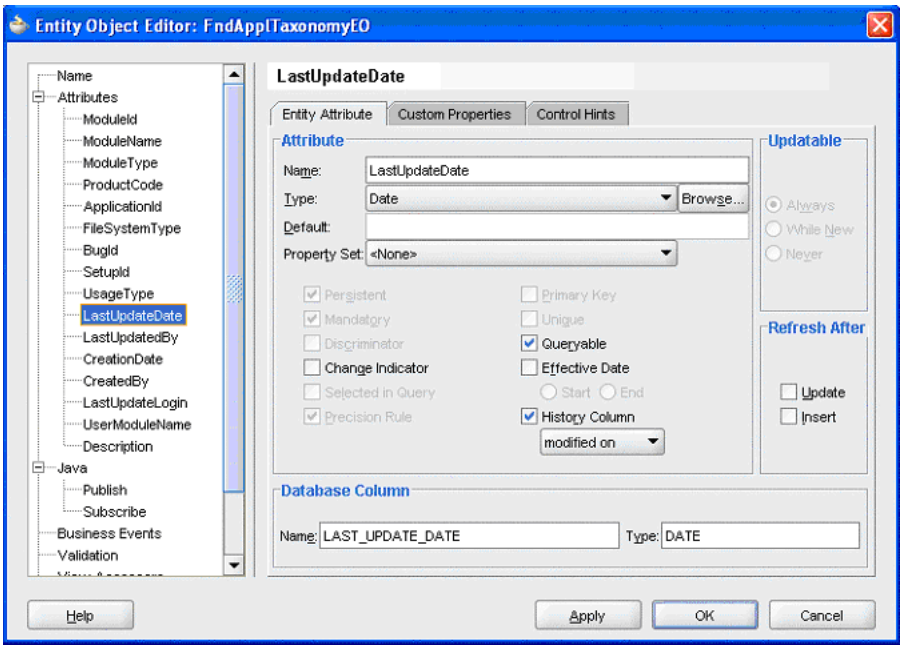
Table 9–1 WHO Column Summary

Column Name	Type	Null?	Description
CREATED_BY	VARCHAR2(64)	NOT NULL	Keeps track of which user created each row.
CREATION_DATE	DATE	NOT NULL	Stores the date on which each row was created.
LAST_UPDATED_BY	VARCHAR2(64)	NOT NULL	Keeps track of who last updated each row.
LAST_UPDATE_DATE	DATE	NOT NULL	Stores the date on which each row was last updated.
LAST_UPDATE_LOGIN	VARCHAR2(32)		Stores the Session ID of the user who last updated the row.

9.3.1 How to Use the Extension

In order for Oracle Fusion Middleware Extensions for Applications to populate your WHO columns automatically, 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.

Figure 9–6 Entity Attribute Wizard: LastUpdateDate



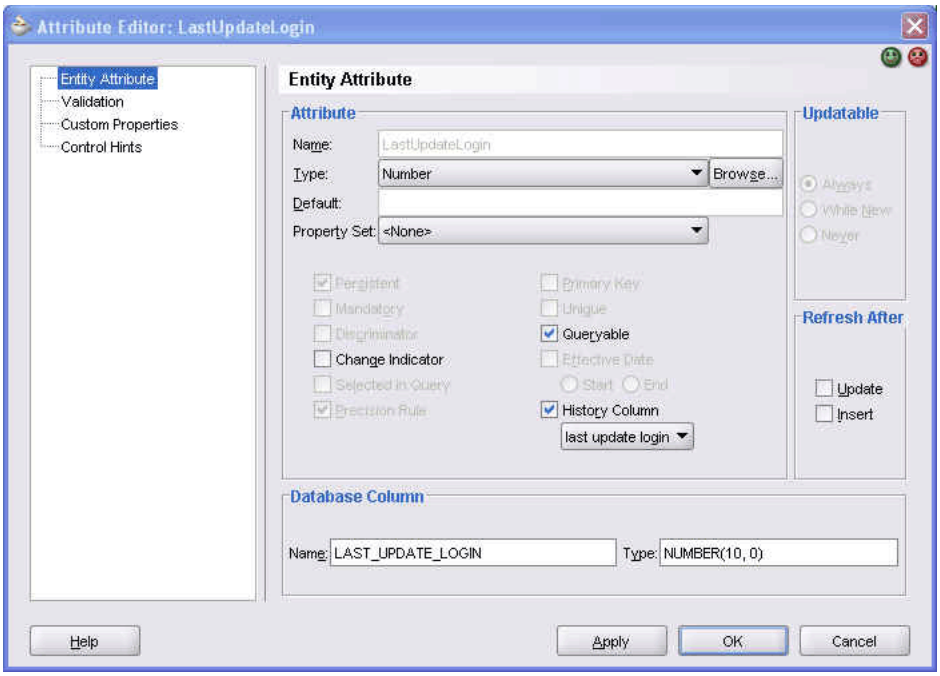
In the example entity, the WHO column **LastUpdateDate** is identified as a `modified on` **History Column** type.

Similarly, identify the following attributes as indicated:

- **LastUpdatedBy** - `modified by`
- **CreationDate** - `created on`
- **CreatedBy** - `created by`

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.

Figure 9–7 Timestamp (java.sql.Timestamp)



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 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 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 How to Use 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

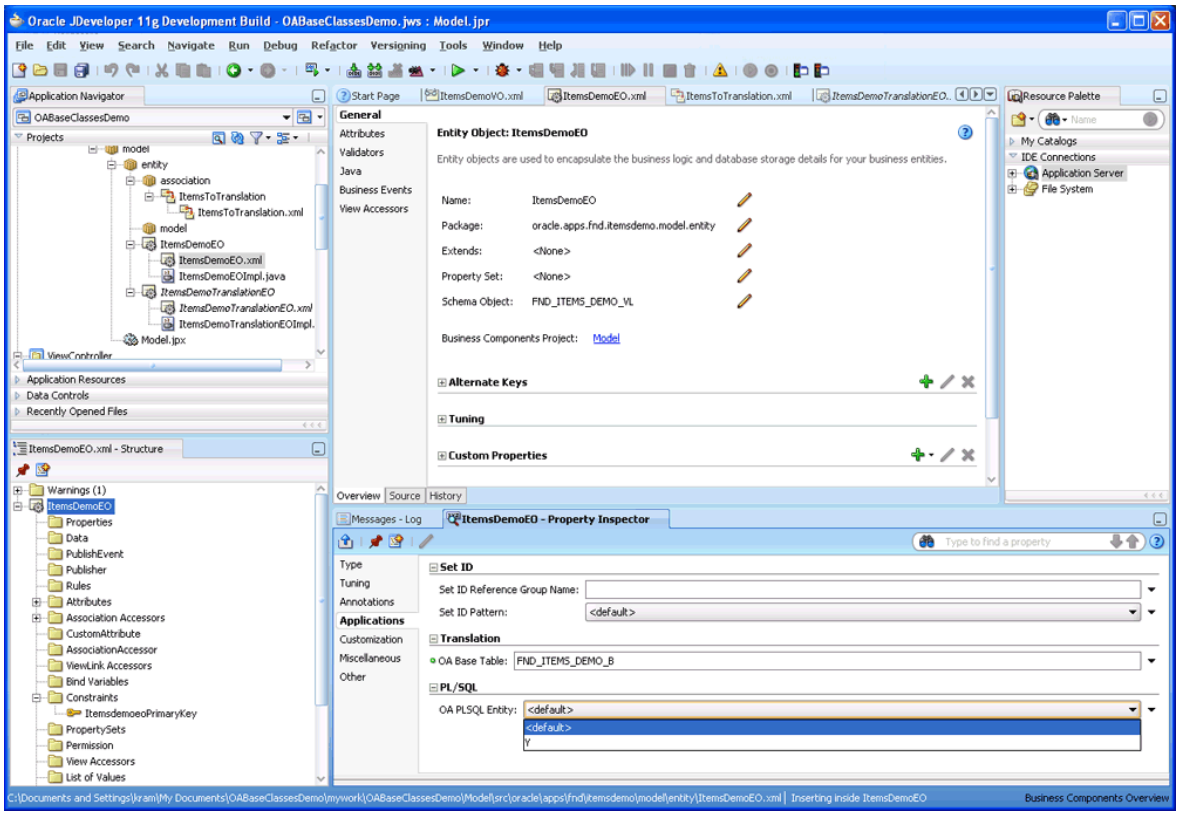
A PL/SQL-based entity object provides an object representation of the data from a table or view and routes the DML operations to stored procedures in the database.

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.

To identify an entity as a PL/SQL one:

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.

Figure 9–8 OA PLSQL Entity Setting



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 Accessing FND Services

Fusion Middleware extensions for Oracle Applications provide the following services:

- Profile
- Lookup
- Message
- Language
- Application
- Taxonomy
- DataSecurity
- Attachments

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 Section 9.5.1, "How to Use the Extension," for implementation information.

9.5.1 How to Use the Extension

The code in Example 9–1 shows how to provide access to an FND service. In this case, it is ProfileService.

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");
```

`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.

9.6 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.

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.

Notes: 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`.

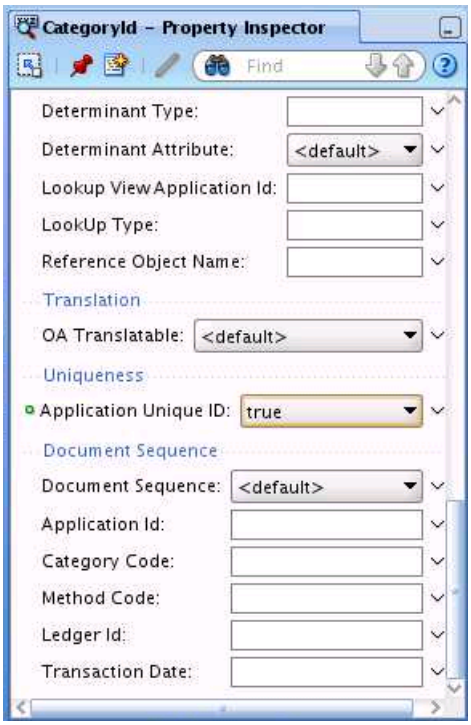
9.6.1 How to Use the Extension

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, 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.

Figure 9–9 Application Unique ID



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:

```
oracle.jbo.server.uniqueid.UniqueIdHelper.getUniqueId(adf.object.unwrapObject());
```

9.7 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 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 Fusion Middleware extensions, see Chapter 11, "Setting Up Document Sequences."

10

Implementing Lookups

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:

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 Oracle Fusion applications.

Lookups Implementation

Once you have completed the development process as discussed in this chapter, and 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 Section 10.6, "Integrating Lookups Task Flows into Oracle Fusion Functional Setup Manager".

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 Section 10.2.1, "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.

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"
```

- 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 Chapter 8, "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.

10.1.2 Standard, Set-Enabled, and Common Lookup Views

All lookups business objects exist in the **publicEntity** subpackage of the **oracle.apps.fnd.applcore.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 the purpose of 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 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

Lookup Code	Description
U	User
E	Extensible
S	System

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 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 Section 10.1.2, "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:

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:

```
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:

```
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:

```
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 Section 10.1.2, "Standard, Set-Enabled, and Common Lookup Views".

10.3 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:

1. Import the lookups standard view objects into your project and make sure they are referenceable.
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 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 Section 10.4.1, "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 Section 10.4.2, "How to Define 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.

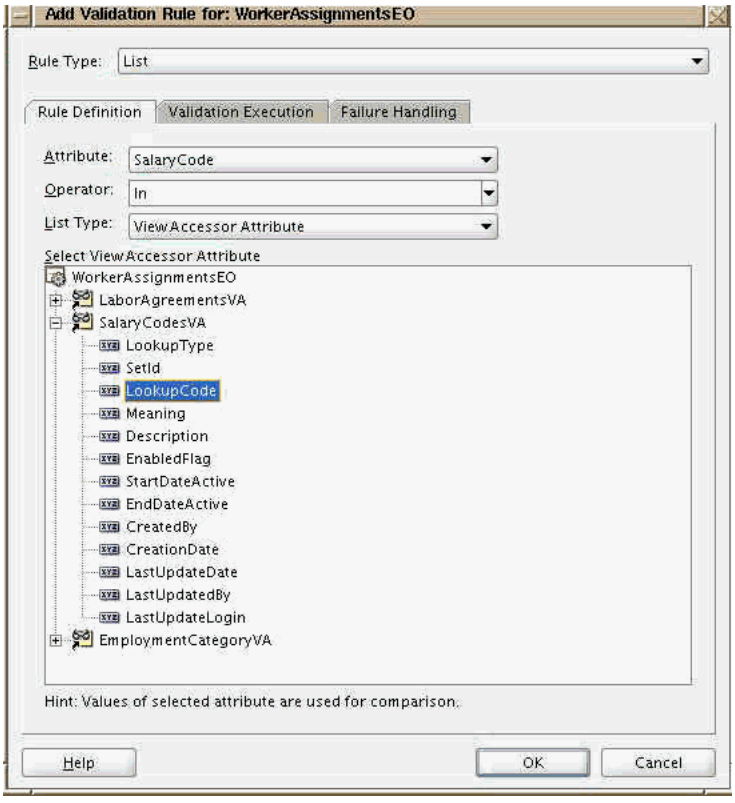
10.4.1 How to Define a List Validator

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:

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`.

Figure 10–1 Lookups List Validator Rule Definition

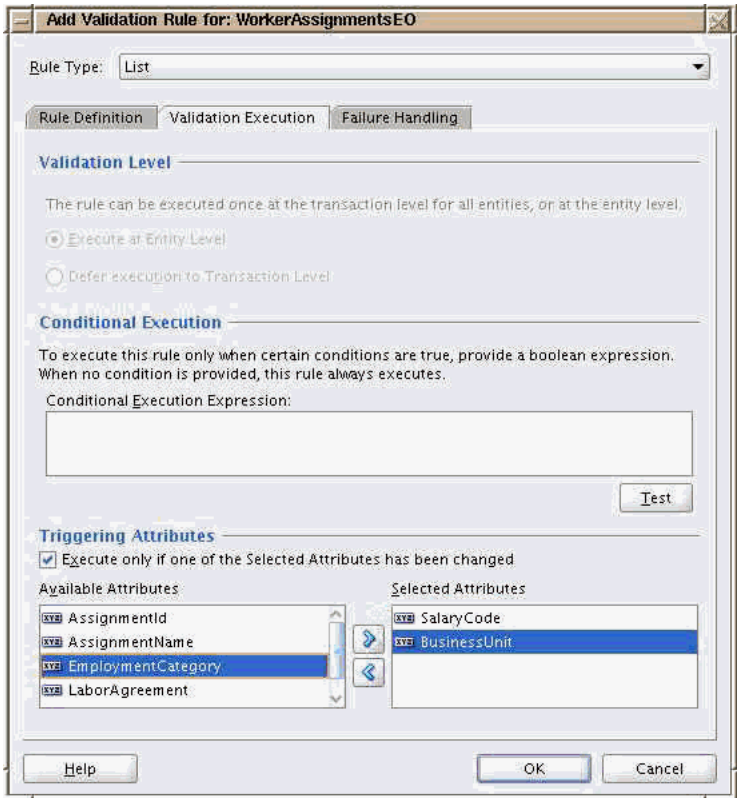


4. On the Rule Definition tab, select the foreign reference column that is the lookup code (for example, `SalaryCode`) as the attribute.
5. Select `In` as the operator.

6. Select `View Accessor Attribute` as the list type.
7. From the Select View Accessor Attribute list, select `LookupCode` as the view accessor validation target lookup code attribute.
8. 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.

9. In the Triggering Attributes section, select the determinant attribute from the left-hand list and shuttle it to the right-hand list.
10. Optionally, on the Failure Handling tab, specify a failure error message.
11. Click **OK** to create the list validator for this lookup.

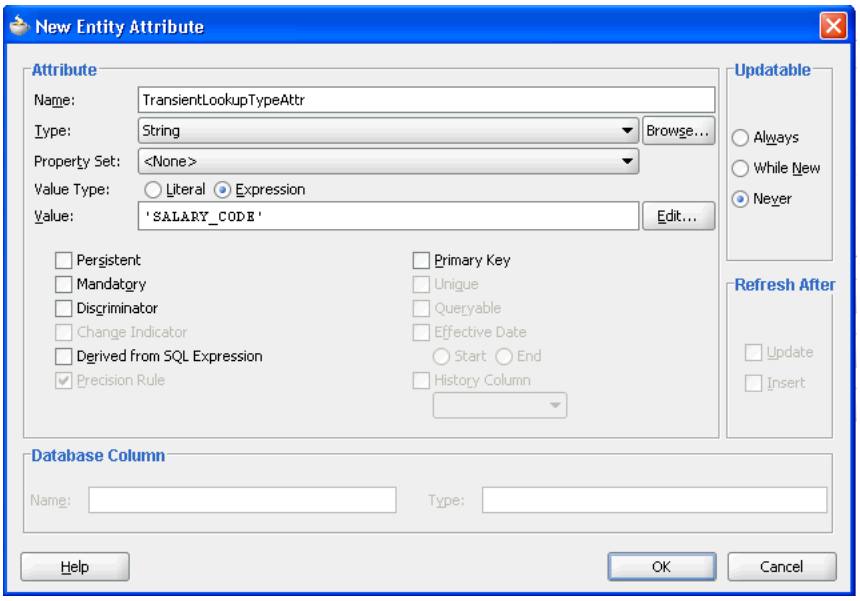
10.4.2 How to Define a Key Exists Validator

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.

To define a transient lookup type:

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

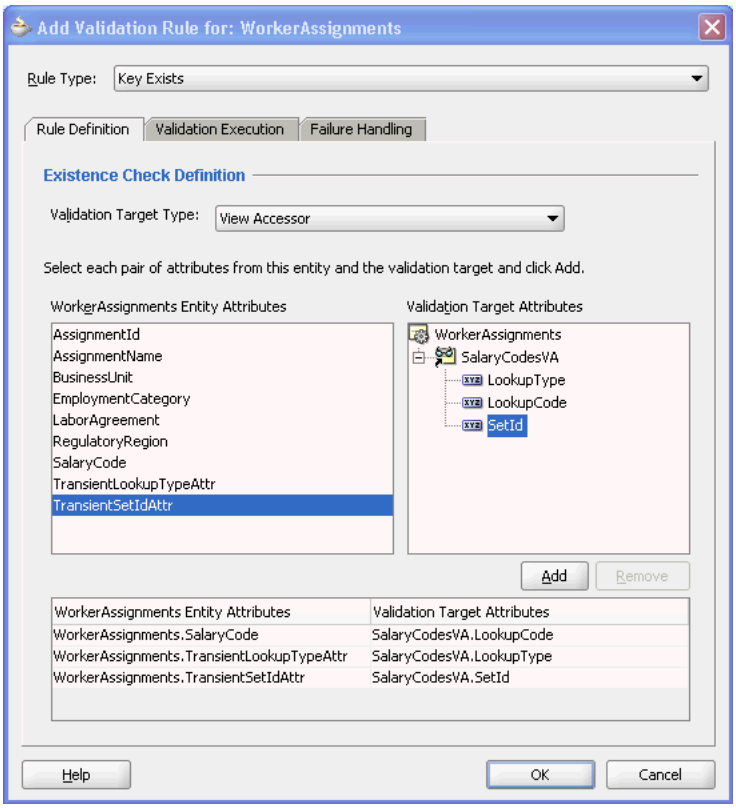


3. Set the **Type** to `String`.
4. Set the **Value Type** to `Expression`, and provide a constant value for the attribute.
5. Deselect the **Persistent** checkbox.
6. In the **Updatable** section, select `Never`, then click **OK** to create the transient attribute.

To create a key exists validator:

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`.

Figure 10–4 Lookups Key Exists Validator Rule Definition



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.

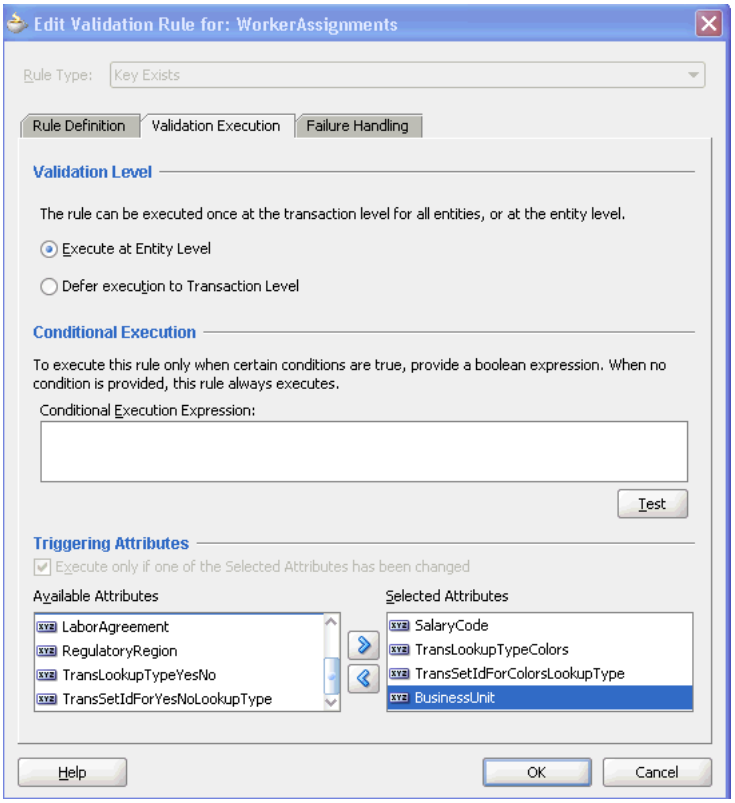
   Click **Add** to include the attribute pair on the mapping list.
5. Select the entity object transient lookup type attribute on the left-hand list, and the corresponding view accessor validation target lookup type attribute on the right-hand list.

   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.

Figure 10–5 Lookups Key Exists Validator Execution



Note: The foreign key attributes (including transient atributes) 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 Message**.
10. Click **OK** to generate the key exists validator for this lookup.
11. Save your project.

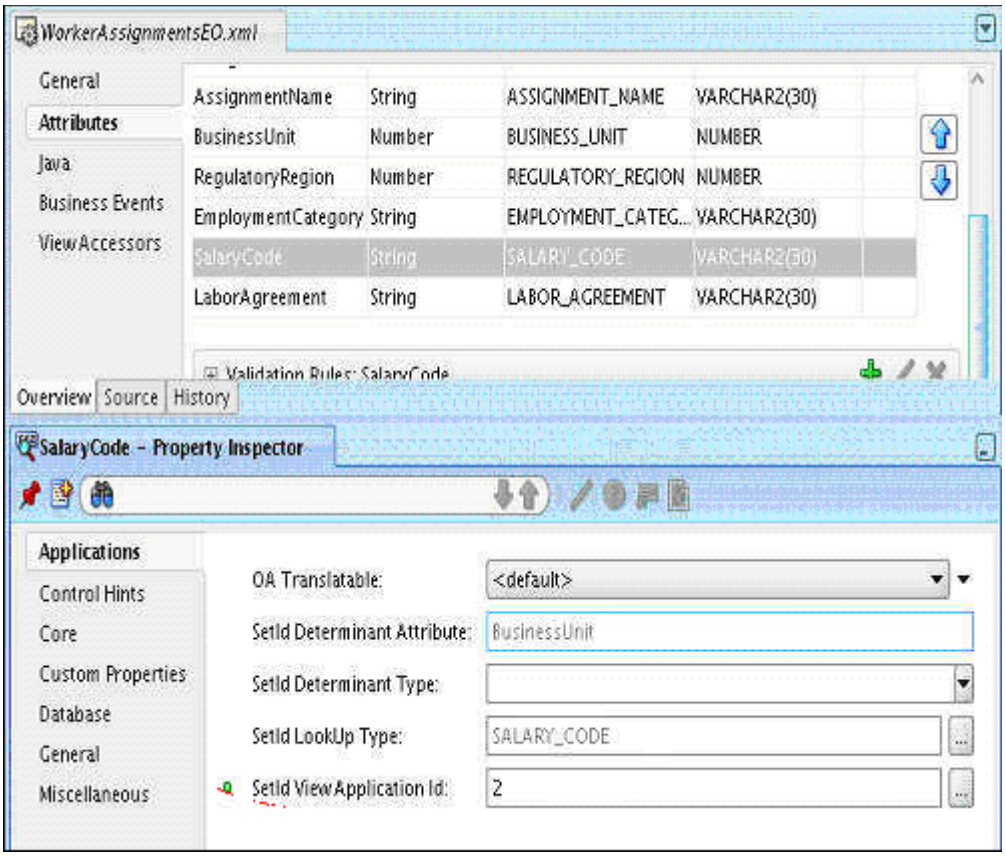
10.5 Annotating Lookup Code Reference Attributes for Set-Enabled Lookups

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.

Figure 10–6 Lookup Type and View Application ID for a Lookup Code Reference



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 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 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. 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

Task Flow Name	Task Flow XML	Parameters Passed	Behavior
Manage Standard Lookups	/WEB-INF/oracle/apps/fnd/applcore/lookups/publicUi/flow/ManageStandardLookupsTF.xml#ManageStandardLookupsTF	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*'	This task flow enables you to create and edit lookups in the centrally owned Standard view (view application = 0).
Manage Set-Enabled Lookups	/WEB-INF/oracle/apps/fnd/applcore/lookups/publicUi/flow/ManageSetEnabledLookupsTF.xml#ManageSetEnabledLookupsTF	To invoke search mode to query and edit lookup types and their codes in the Set Enabled Lookups view: mode='search' To restrict search mode to Set Enabled 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*'	This task flow enables you to create and edit lookups in the centrally owned Set Enabled view (view application = 2).

Table 10–2 (Cont.) Lookups Task Flows and Parameters

Task Flow Name	Task Flow XML	Parameters Passed	Behavior

Manage Common Lookups	`/WEB-INF/oracle/apps/fnd/applcore /lookups/publicUi/flow/ ManageCommonLookupsTF.xml# ManageCommonLookupsTF`	To invoke search mode to query and edit lookup types and their codes in the Common Lookups view:
		`mode='search'`
		
		To restrict search mode to Common 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'`
			This task flow enables you to create and edit lookups in the centrally owned Common Lookups view (view application = `3`).

For more information about task flows, see the *Oracle Fusion Applications Common Implementation Guide*.

11

Setting Up Document Sequences

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:

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.

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 11–1 Document Sequence Terminology

Term	Description
Document Sequences	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.
Document Sequence Categories	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).
Sequence Assignments	The user assigns a sequence for each category. The assignments are owned by a set of books and are effective within a date range. Manual document entry through a form and automatic document creation through a batch process can have separate sequence assignments. Currently, legal entities for the same set of books must share document sequences. If each legal entity requires its own numbering sequence, a separate set of books must be created for each legal entity.

11.2 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 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 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 11–2.

Table 11–2 Audit Table Columns and Types

Name	Null?	Type
DOC_SEQUENCE_ID	NOT NULL	NUMBER(18)
DOC_SEQUENCE_VALUE	NOT NULL	NUMBER(15)
DOC_SEQUENCE_ ASSIGNMENT_ID	NOT NULL	NUMBER(18)
CREATION_DATE	NOT NULL	TIMESTAMP(6)
CREATED_BY	NOT NULL	VARCHAR2(64 CHAR)
LAST_UPDATE_DATE	NOT NULL	TIMESTAMP(6)
LAST_UPDATED_BY	NOT NULL	VARCHAR2(64 CHAR)
LAST_UPDATE_LOGIN		VARCHAR2(32 CHAR)
ENTERPRISE_ID	NOT NULL	NUMBER(18)

11.5 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.

11.5.1 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, Timestamp txnDate, 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 in order 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

```
/**
  * Override of EntityImpl.postChanges() to handle document sequencing.
```

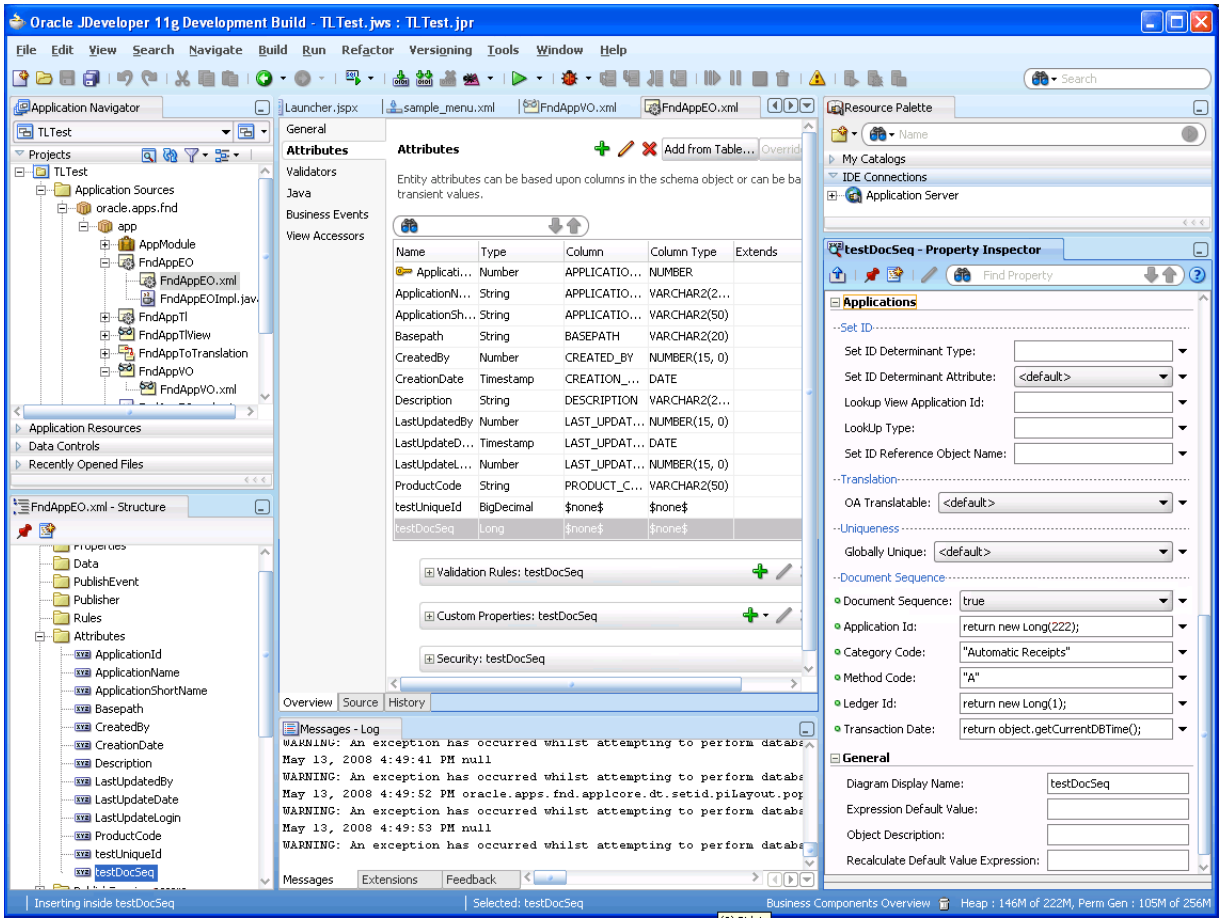
```
   * If an entity attribute has been identified that it should be populated
   * using a document sequence (in the Applications Property Inspector panel),
   * then at this point in the entity life cycle, we will populate the attribute
   * with a document sequence based on the inputs, provided the 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_SET_OF_BOOKS_ID (Ledger Id),
   * 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 txnDate Document 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_SET_OF_BOOKS_ID (Ledger Id),
     * 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 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)
    {...}
```

11.5.1.1 What Happens with Document Sequences at Design Time

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 11–3 lists the metadata fields in the Property Inspector window and their descriptions.

Table 11–3 Additional Metadata

Field	Description
Application Id	The application ID. The Groovy expression should return an object of type `Long`.
Category Code	The document sequence category code. The Groovy expression should return an object of type `String`.
Method Code	The document sequence method code. Select from the following: `"A"` (Automatic), `"M"` (Manual), or `null` (both modes). The Groovy expression should return an object of type `String`.
Ledger Id	The ledger ID to use for this document sequence. The Groovy expression should return an object of type `Long`.
Transaction Date	The document transaction date. The Groovy expression should return an object of type `Timestamp`.

11.5.1.2 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 when invoked in manual mode

11.6 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

```
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
  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

```
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

```
declare
  ret number;
  docseq_val number;
  docseq_id number;
begin
  -- Verify a sequence value for an 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
           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('Value '||to_char(docseq_val)||' is valid');
  end if;
end;
```

11.7 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 11–4 lists the task flows and their parameters.

Table 11–4 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.xml#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 more information about task flows, see *Oracle Fusion Applications Common Implementation Guide*.

12

Implementing Audit Trail Reporting

This chapter describes how to implement Audit Trail Reporting in Oracle Fusion Applications.

This chapter contains the following sections:

12.1 Introduction to Audit Trail Reporting

Audit Trail is a history of the changes that have been made to data in Oracle Fusion Applications. Audit Trail includes information such as who has accessed an item, what operation was performed on it, when it was performed, and how the value was changed. The audit information is logged without any interaction from the end user.

12.2 Implementing Audit Trail Reporting

Database auditing involves observing a database to be aware of the actions of database users. This is often for security purposes, such as to ensure that information is not accessed by those without the permission to access it. 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 DML actions on this object, that is, its underlying database schema objects, are captured and stored for reporting purposes.

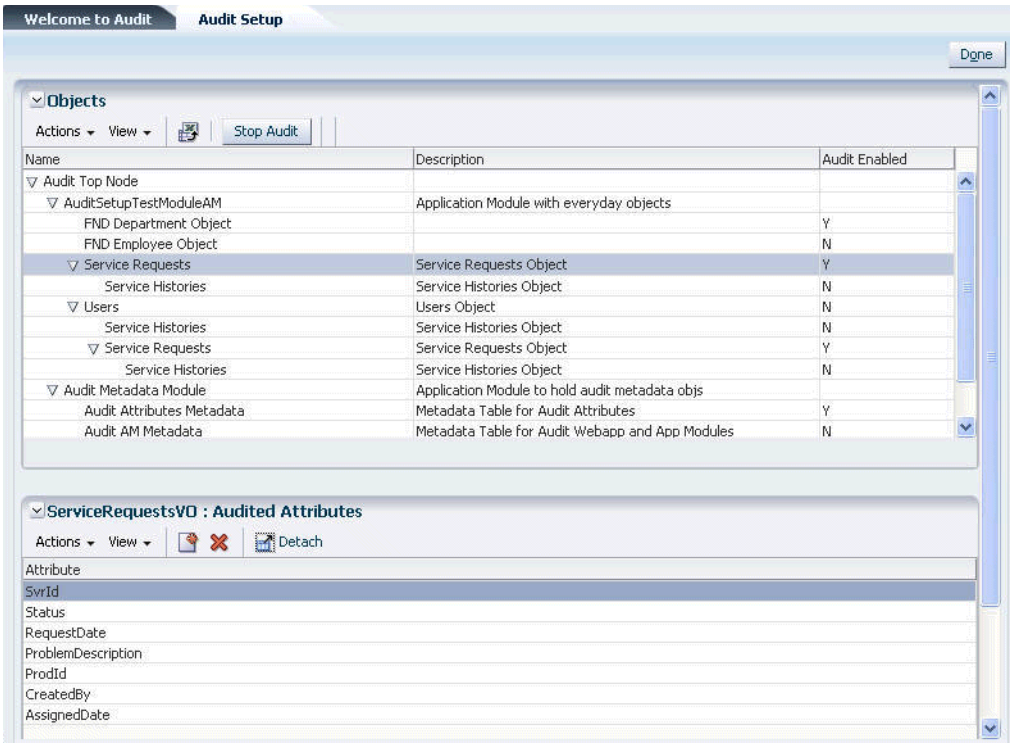
The Audit Setup UI page is a public taskflow that is made available through the Oracle Middleware Extensions for Applications library. This taskflow can be included in an application that is accessible by the administrator. The UI consists of an Applications Tree Table that holds the structure of the 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.

The Audit application holds all code for the Audit Setup and Audit Reporting taskflows.

The user interface, shown in Figure 12–1:

- Allows an administrator to select the objects and its specific attributes to audit.
- Provides the ability to stop auditing.
- Displays which objects currently are being audited.

Figure 12–1 Audit Setup Page



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.

WEBAPP	APPLICATION_MODULE
VARCHAR2(80)	VARCHAR2(240)

- FND_AUDIT_ATTRIBUTES holds seed data for the applications modules that can be audited for a given application. This table holds data of the attributes in a view object that have auditing enabled. This is an internal table and should not be changed.

WEBAPP	VIEW_ OBJECT	VIEW_ ATTRIBUTE	ENABLED_ FLAG	TABLE_ NAME	COLUMN_ NAME
VARCHAR2 (80)	VARCHAR2 (240)	VARCHAR2 (80)	VARCHAR 2(1)	VARCHAR2 (30)	VARCHAR2 (30)

Important Code Components

The important code components involved in creating the user interface are:

- **AuditSetup.jsff:** The page fragment that holds the components for the Audit Setup UI. It consists of the Applications Tree Table components containing the structure of the applications modules from the `fnd_audit_webapp_am` for the given application and its children.

- **AuditTreeBean.java:** Session scoped managed bean class for the AuditSetup page fragment. It contains the action/action listener, binding, and display methods. It also creates the Tree Model structure for the Applications Tree Table and filters the data shown in the Applications Table for the selected view object.
- **AuditModelUtility.java:** Utility file that holds the logic for reading the applications modules and traversing through the structure to show those objects that are auditable.

12.2.1 How to Set Metadata

This section covers the steps needed to populate and define metadata needed to run the Audit taskflows.

To populate and define metadata:

1. Update the ApplicationAuditDB connection to point to your Audit Vault Server (login as auditor) in your application.
2. Make sure every application has the `webApp` defined and set. This is done by populating the ASK tables.
3. Design from where the audit setup flow and reporting flow will be launched. This will be a link. It can be in the tasklist, or within the main area of a page. For example the administration flows can be as tasks in the region area, whereas the business object specific reporting can be placed beside the components that display this business object, such as a table.

   This link will be protected to be visible to only those with the correct privileges.
4. 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.
5. Determine which business objects of each application module may be audited. Once the business objects are identified, set a non-translatable custom property 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.

```
<Properties>
    <CustomProperties>
      <Property
        Name="Auditable"
        Value="Y"/>
    </CustomProperties>
</Properties>
```

6. Determine which attributes of each business object to not audit. By default, all attributes are shown *except*:
   - Attributes with a non-translatable custom property with Name="Auditable" and Value="N". This follows an opt-out policy. That is, if no property is defined, the value is assumed to be "Y."
   - Attributes that have display hint set to hidden.
   - Attributes that are history columns; for example: Created By, Creation Date, Last Updated By

```
<ViewAttribute
    Name="Attribute1"
```

```
    IsUpdateable="false"
    PrecisionRule="true"
    EntityAttrName="Attribute1"
    EntityUsage="AttributeEO"
    AliasName="ROWID">
    <Properties>
      <CustomProperties>
        <Property
          Name="Auditable"
          Value="N"/>
      </CustomProperties>
      <SchemaBasedProperties>
        <DISPLAYHINT
          Value="Hide"/>
      </SchemaBasedProperties>
    </Properties>
  </ViewAttribute>
```

7. To have a user-friendly display name for all application modules, business objects and attributes names the display name property for each needs to be set. For applications modules and business objects it is also a good idea to put in a description, since the description will be shown in the Audit Setup UI and also will give the administrator further information on the business objects and/or application module. The display name and description are available as a property.

```
<Properties>
    <CustomProperties>
      <Property
        Name="Auditable"
        Value="Y"/>
    </CustomProperties>
    <SchemaBasedProperties>
      <LABEL
        ResId="ViewObjectVO_LABEL"/>
      <TOOLTIP
        ResId="ViewObjectVO_TOOLTIP"/>
    </SchemaBasedProperties>
  </Properties>
  <ResourceBundle>
    <XliffBundle

id="oracle.apps.fnd.applcore.audit.test.model.view.common.ViewObjectVOMsgBundle
"/>
  </ResourceBundle>
```

8. For custom business objects, identify the application module the custom business objects belong to, and add this to the metadata table as defined in step 4. Define the "Auditable" property as defined in step 5 on the custom business objects.

12.2.2 How to Use Audit Taskflows

This section covers how to use the Audit taskflows and their taskFlowIDs.

Audit Setup Taskflow - For Administrator Use Only

Product teams should create a control flow for the Done outcome to go back to the original page from which the setup page was accessed. To use the public taskflow for Setup, the package structure is `/WEB-INF/oracle/apps/fnd/applcore/audit/ui/flow/AuditSetupAdminTF.xml#AuditSetupAdminTF`. This flow provides the **Done** button.

```
<itemNode id="auditSetup" focusViewId="/AuditPage" label="Audit Setup"
     taskType="dynamicMain"

taskFlowId="/WEB-INF/oracle/apps/fnd/applcore/audit/ui/flow/AuditSetupAdminTF.xml#
AuditSetupAdminTF"/>
```

Audit Reporting Taskflow - For Administrator Use Only

To use the public taskflow for "Admin reporting" the package structure is `/WEB-INF/oracle/apps/fnd/applcore/audit/ui/flow/AuditReportAdminTF.xml#AuditReportAdminTF`.

```
<itemNode id="auditObjectReport" focusViewId="/AuditPage"
          taskType="dynamicMain"

taskFlowId="/WEB-INF/oracle/apps/fnd/applcore/audit/ui/flow/AuditReportAdminTF.xml
#AuditReportAdminTF" label="Admin UI"/>
```

Audit Reporting Taskflow Business Object Specific

To use the public taskflow for "Object specific reporting" the package structure is `/WEB-INF/oracle/apps/fnd/applcore/audit/ui/flow/AuditReportObjectSpecificTF.xml#AuditReportObjectSpecificTF`.

These parameters need to be passed:

- Full name of the view object (Required)
- Primary Key (Required)
- FromDate (Optional)
- ToDate (Optional)

```
<itemNode id="auditAdminReport" focusViewId="/AuditPage"
          taskType="dynamicMain"

taskFlowId="/WEB-INF/oracle/apps/fnd/applcore/audit/ui/flow/AuditReportObjectSpeci
ficTF.xml#AuditReportObjectSpecificTF"
          label="Object Specific UI"

parametersList="viewObject=oracle.apps.fnd.applcore.patterns.test.model.view.Servi
ceRequestsVO;pkValue=157;fromDate=01/01/2010;toDate=12/31/2010"/>
```

Part III

Defining User Interfaces

This part of the Developer's Guide discusses some of the Oracle Application Development Framework (Oracle ADF) user interface features that you can incorporate into your Oracle Fusion Applications.

The *Getting Started with your Web Interface* chapter provides information about how to create a page and what the wizard settings should be. It also presents the basic information that is necessary before creating the Application User Interface.

The *Implementing the UI Shell* chapter provides information about the UI Shell and Navigator Menu components used to implement user interface features in JDeveloper. The UI Shell is a page template whose contents are determined by the menu metadata held in the Navigator Menu.

The *Implementing UIs in JDeveloper with Applications Tables, Trees, and Tree Tables* chapter discusses the Applications Tables, Trees and Tree Tables components used to implement user interface features in JDeveloper. *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. The *Applications Tree* component provides the basic capabilities that satisfy the requirements specified in the Application UX designs. These include tree toolbar with default buttons, facets for adding ADF tree, custom toolbar buttons, and so on, and default implementations for tree actions. 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 Applications UX standards.

The *Implementing Applications Panels, Master-Detail, Hover, and Dialog Details* chapter discusses the Applications Panels, Master-Detail, Detail on Demand, and Dialog Details components used to implement user interface features in JDeveloper. *Applications panels* help you create specific UI components as part of the UI Applications patterns. You must use Applications panels to standardize layout and appearance for all your page forms and buttons, including read-only pages. 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. *Dialog details* are appropriate for use when information needs to be accessed quickly and then dismissed. The details are shown in a modeless dialog window.

Implementing Skinning describes how to change the look and feel of your application by changing the skin, without changing the content. The chapter deals specifically with skinning as applied to Oracle Fusion applications and the UI Shell.

The *Implementing Attachments* chapter provides guidelines for implementing Attachments at design time in a quick and simple manner using Oracle Fusion Middleware components. The Attachment component provides a declarative and

simple programming mechanism for you to add attachments to the UI pages that you create for web applications. Once 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.

The *Organizing Hierarchical Data with Tree Structures* chapter describes how to create, update, and delete tree structures, trees, and tree versions, and how to develop applications using trees. Oracle Fusion tree management allows data in Oracle Applications to be organized into a hierarchical fashion, and allows Oracle Applications customers to create tree hierarchies based on their specific data.

The *Working with Localization Formatting* chapter describes the Oracle applications standards and guidelines for working with 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 JDeveloper and ADF, localization requires implementing formatting patterns so as to properly display the data according to local standards.

This part contains the following chapters:

13

Getting 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:

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 Web User Interface Developer's Guide for Oracle Application Development Framework*.

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

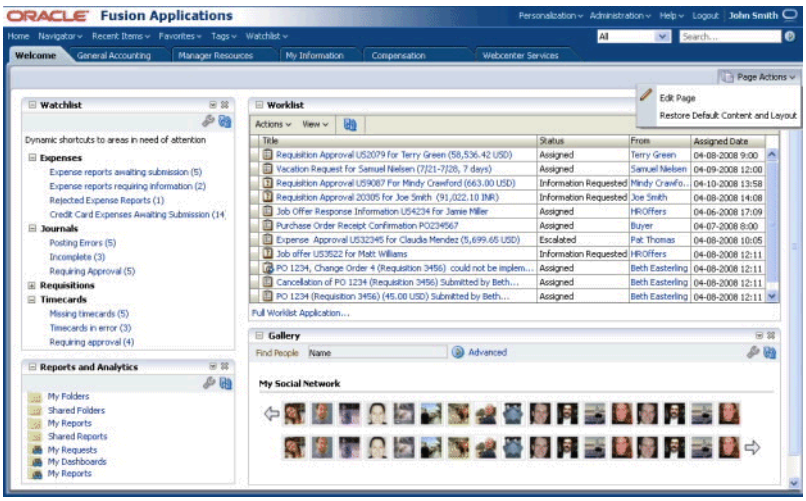
Dashboards and Workareas define the basic structure of a page. Oracle Fusion Guidelines, Patters and Standards (GPS) defines a set of design patterns. Design

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.

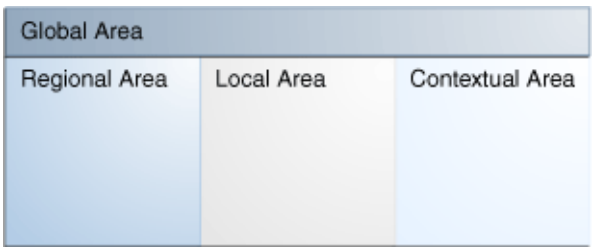
Figure 13–1 Home Based Dashboard Example



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 Workareas 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, in conjunction 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 Chapter 14, "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 needs to 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 Web User Interface Developer's Guide for Oracle Application Development Framework*.

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:

- Chapter 15, "Implementing UIs in JDeveloper with Application Tables, Trees and Tree Tables,"
- Chapter 16, "Implementing Applications Panels, Master-Detail, Hover, and Dialog Details," and

14

Implementing the UI Shell

This chapter discusses the UI Shell page template used to build web pages, and the components used to implement user interface features in JDeveloper, such as menus, task flows, security, search, navigation, and the home page UI.

This chapter includes the following sections:

For basic information and detailed information of the features, see:

- Chapter 16, "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 also 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 since it is loaded dynamically.

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 Middleware alone.

Before you begin:

You should be familiar with JDeveloper, be able to create and run JSF pages, and be able to create an Oracle Application Development Framework (Oracle ADF) task flow.

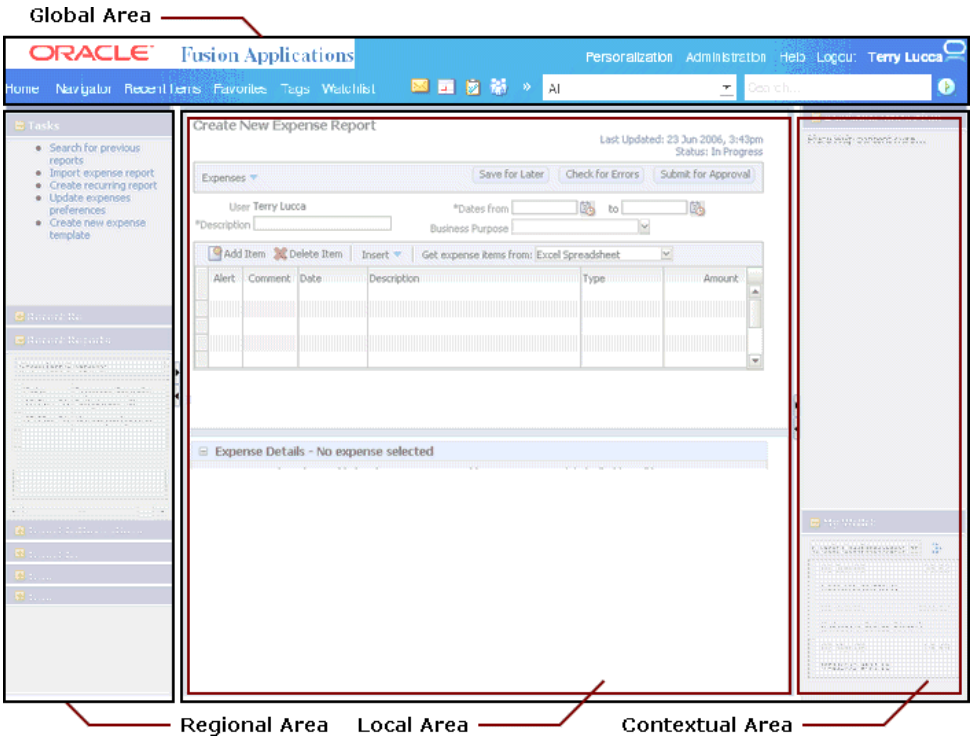
14.1.1 Standard Related to UI Shells

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, mandatory areas: global, regional, local, and contextual, as shown in Figure 14–1.

Figure 14–1 UI Shell Areas



- The shell is optimized for a screen resolution of 1280x1024 pixels.
- The four areas are:
  - **Global Area**: The global area, across the full width at the top of the UI shell, is stable, consistent, and persistent for an individual user. It contains controls that, in general, drive the contents of the other three areas. See Section 14.1.2.1, "Global Area Standard Links."
  - **Regional Area**: The regional area is in the left pane of the UI shell. It has controls and content that, in general, drive 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.
  - **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, in general, drive the contents of the contextual area.
    - **Main Area**: This term designates the combination of the Local Area and the Contextual Area.
  - **Contextual Area**: The contextual area is in the right pane of the UI shell, with controls and contents that, in general, are driven by controls in, or the content or state of, the local area; although in specific cases the contextual area can also, in turn, drive the contents of the local area (causing a local-area reload).
- Application designers, customers, and administrators can set the regional area 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 shall be able to bind contextual area content to the local area such that each invocation of a local area shall automatically cause a relevant contextual area in the right state to show up alongside the local area.

14.1.2.1 Global Area Standard Links

The Global Area incorporates a number of built-in indicators and links.

- **Home**

  Click this link to return to the defined Home page. See Section 14.18, "Implementing the Oracle Fusion Home Page UI."

- **Navigator**

  The Navigator menu, shown in Figure 14–18, is rendered when the Navigator link is clicked on the UI Shell. See Section 14.5.1.2, "Displaying the Navigator Menu."

- **Recent Items**

  Recent Items tracks a list of the last *20* task flows visited by a user. See Section 14.11, "Implementing Recent Items."

- **Favorites**

  Add to Favorites takes the most recent Recent Item (see Section 14.11, "Implementing Recent Items") and persists it into the Favorites list.

- **Tags**

  Tagging is a service that allows users to add tags to arbitrary resources in Oracle Fusion Applications so they may collectively contribute to the overall taxonomy and discovery of resources others have visited. See Section 14.10, "Implementing Tagging Integration."

- **Watchlist**

  Watchlist is a user-accessible UI that provides a summary of items the user can track with drilldown shortcuts. See Section 14.12, "Implementing the Watchlist."

- **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. See Section 14.13, "Implementing Group Spaces."

- **Personalization**

  The Personalization menu options let you set your preferences, edit the current page, and reset the content and layout. See Section 14.6, "Using the Personalization Menu."

- **Accessibility**

  The Accessibility link appears on pages that can be accessed without logging in. It will allow users to set their accessibility preferences because the Personalization menu, which includes preferences, is hidden for anonymous users.

- **Administration**

  The Administration menu options allow you to customize the current page at a multi-user level, allows you to manage sandboxes, and allows you access to the setup applications. See Section 14.8, "Using the Administration Menu."

- **Help**

The Help menu options let you control trace levels, run diagnostics, and provide an About page that lists information about the application. See Section 14.9, "Using the Help Menu."

- **Sign In / Sign Out**

  Note: When signing in, users *always* are directed to the application's home page.

  There are two possible scenarios during run-time:

  - 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.
    - There is no user name displayed next to the commandLink.
  - The application is secured with ADF Security.

    If the user is Logged In:

    - The commandLink displays the text Sign Out and is enabled.
    - The logged-in user name is displayed next to the Sign Out link.
    - If Oracle WebLogic Server is configured to authenticate with Oracle Internet Directory (OID) 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 Section 14.7, "Implementing 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.
    - No user name is displayed next to the Sign In link.

When the **Sign Out** link is clicked, the page is always redirected to the application's 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.

14.2 Populating a UI Shell

The UIShell 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 a menu metadata. This informs it of which taskflows to load and where. The UIShell 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 UIShell template follow the normal ADF Security framework.

When you create an application using the Fusion Web Application (ADF) template, two projects automatically are created for you: the data 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.

14.2.1 How to Create a JSF Page

Creating a page 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

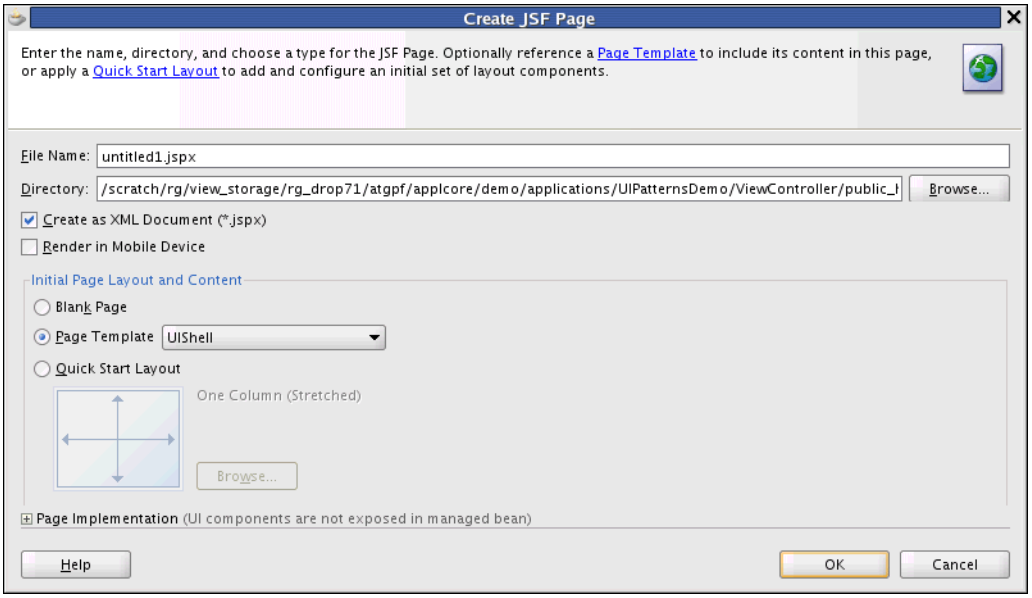
For more information about starting JDeveloper, see Chapter 2, "Setting Up Your Development Environment."

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 14–2.

Figure 14–2 Create JSF Page Dialog



3. In the dialog:
   - Enter a file name and directory path.

     The filename should follow these patterns:

     – [<Product Code><LBA Prefix>]<Role>Dashboard.jspx

     – [<Product Code><LBA Prefix>]<Object>Workarea.jspx
   - From the **Use Page Template** list, select **UIShell**.

Note: The `af:skipLinkTarget` tag has been incorporated in the UIShell.jspx template so developers do not need to code this required accessibility feature in each page.

Specifically, `<af:skipLinkTarget/>` has been inserted before the SingleObjectContextArea facet in UIShell.jspx:

```
<af:panelGroupLayout inlineStyle="width:100%;"
id="soContextParent">
     <af:skipLinkTarget/>
        <af:facetRef facetName="SingleObjectContextArea"/>
</af:panelGroupLayout>
```

- Check **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 for dynamic loading of task flows at runtime. It also creates the Navigator Menu and Task List Menu. See Section 14.5, "Working with the Global Menu Model" and Section 14.2.1.1, "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 Section 14.2.1.1, "Working with the Applications Menu Model" for how to add a menu to the page.

Table 14–1 itemNode Properties of a JSF Page

ItemNode Property	Property Value	Description
`action`	Name that has been assigned to the action.	Navigate to the page defined by the action.
`dataControlScope`	string	Values are **shared** (the default) or **isolated**. This is set at the page level itemNode. When dataControlScope is set to "isolated", the UI Shell loads the Main Area and Regional Area task flows with dataControlScope set to "isolated". When dataControlScope is set to "shared", the UI Shell loads the Main Area and Regional Area task flows with dataControlScope set to "shared". For example: `<itemNode id="itemNode_AppsPanelTests_TabsWA" label="label_AppsPanelTests_TabsWA" action="AppsPanelTests_TabsWA" focusViewId="/AppsPanelTests_TabsWA" dataControlScope="isolated">`
`isDynamicTabNavigation`	`True` or `False`	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="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. Multiple **defaultMain** definitions are allowed and will open multiple tabs on page load. The first one with `disclosed="true"` will be the tab in focus. If the property value is not defined, it defaults to true.
`id`	Unique identifier.	

Table 14–1 (Cont.) itemNode Properties of a JSF Page

ItemNode Property	Property Value	Description

`label`	string	What appears in the work area title.
		Note: For all UIShell work area pages with Data Visualization Tool (DVT) components in the default Main 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 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 UIShell pages with DVT components in their dynamic Main flows, the title is set on the `AdfRichDocument` by UIShell code for `openMainTask`, `closeMainTask` and tab switch.

`focusId`	Name of the View Activity.	
`formUsesUpload`	True or False. Default is False.	To set the UI Shell's af:form uses Upload value to "true," product teams need to add the `formUsesUpload="true"` property to the itemNode that represents the JSPX (similar to the way isDynamicNavigation is set.)
`regionalAreaWidth`	Numeric value	See Section 14.4.2, "How to Control Regional Area Task Flows".
`isRegionalAreaCollapsed`	True or False	See Section 14.4.2, "How to Control Regional Area Task Flows".

14.2.1.1 Working with the Applications Menu Model

Page and task flow information are local to a particular JDeveloper application or project and are exposed using the Applications Menu Model.

An Applications Menu is related to a local JSPX file and includes the tasks list, defaultMain, and defaultRegional. A menu is created for each J2EE application.

14.2.1.1.1 How to Create an Applications Menu To create an ADF menu to access page elements through the Navigator menu 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 Application 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 `adfc-config.xml` is PageA, and the menu file name is PageA_taskmenu.xml. This step also should add the ApplicationsMenuModel managed bean entry into

adfc-config.xml. The managed bean entry should not have the topRootModel managed bean property set. Example 14–1 shows a sample of the generated content in PageA_taskmenu.xml.

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 def 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/Tasks
List.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>
```

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 it in the JDeveloper editor.

   The adfc-config.xml file is located in the following location: ***<project_name>*** > **WEB INF**.

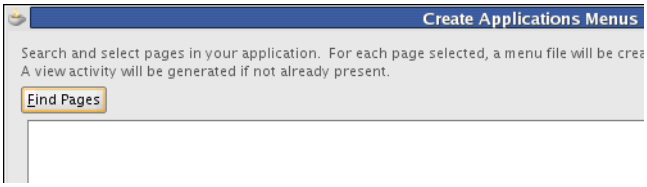
2. Drag pages from the Application Navigator panel to the adfc-config.xml file in the editor.

   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.

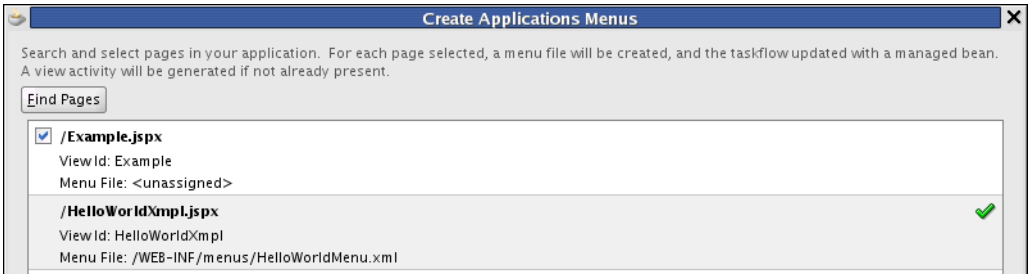
3. Right-click the adfc-config.xml file and choose **Create Applications Menus**. An empty panel, as shown in Figure 14–3, displays.

Figure 14–3 Initial Create Applications Menus Display



4. Click **Find Pages** to populate the display with all the JSPX pages that have been added to `adfc_config.xml`. As shown in Figure 14–4, 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 14–4 Populated Create Applications Menus Display



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 `Example.jspx` 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.

14.2.2 How to Add Default Main Area Task Flows to a Page

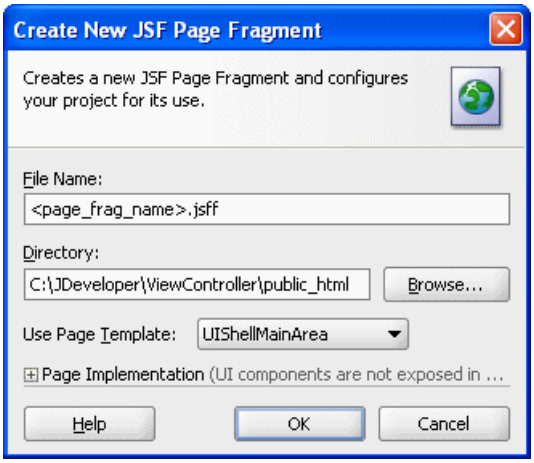
When a user opens an application, he or she expects something to be displayed automatically. A task flow in the default main area accomplishes this.

To add a task flow to the default main area:

1. Choose **File > New > Web Tier > JSF > JSF Page Fragment.**

   The Create JSF Page Fragment dialog shown in Figure 14–5 is displayed.

Figure 14–5 Create New JSF Page Fragment Dialog



2. In the Create New JSF Page Fragment dialog:

   a. Enter a page-fragment name. For example, you might enter `def_main.jsff`.

      The filename should follow these patterns:

      - `<Object><Function>.jsff`
      - `<Object>.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/create pages, just the object should be used (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**.

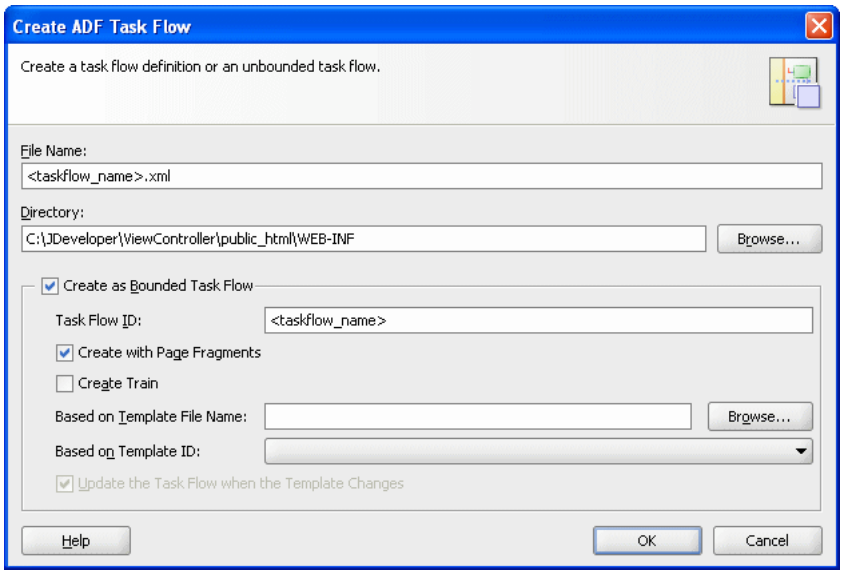
   The local and contextual area facets of the page fragment appear in the editor.

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, "UI Shell Areas" shows 10 task flows.

   The Create ADF Task Flow dialog shown in Figure 14–6 is displayed.

Figure 14–6 Create ADF Task Flow Dialog



5. Make sure 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 that resembles `<f:facet name="localArea"/>`.

   c. In the Application Navigator pane, 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 displays, select **Region**.

      The `<f:facet name="localArea"/>` changes to `<f:facet name="localArea">` and code resembling that shown in Example 14–2 will be inserted after it.

Example 14–2 Creating Region localArea Facet Added Code

```
<af:region value="#{bindings.EmpCreateUpdateFlow1.regionModel}"
                id="EmpCreateUpdateFlow1"/>
</f:facet>
```

      In the Structure window, an `af:region` entry is added following the `f:facet - localArea` entry.

   f. **Note:** This step is *optional*. If you do not need a `contextualArea`, skip to Step 6.

      In the Structure window, select `f:facet - contextualArea`.

   g. In the **Component Palette**, select **ADF Faces > Layout**.

   h. Click **Panel Accordian**. An `af:showDetailItem` entry is created automatically under `af:panelAccordian`.

i. In the Source view, find and highlight the new `<af:showDetailItem ...>` entry.

j. In the Application Navigator pane, select an applicable task flow, such as the one you created in Step 4, and drag and drop it onto the highlighted entry in Source view.

k. From the **Create** menu that displays, select **Region**.

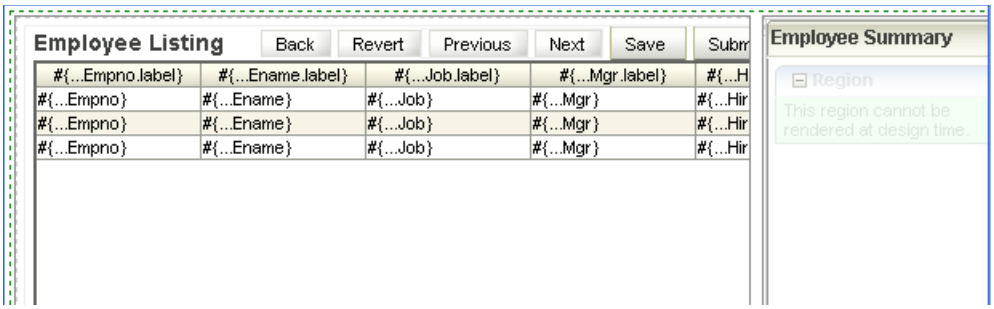
l. Click **OK** on the Edit Task Flow Binding dialog that displays.

Code resembling that shown in Example 14–3 will be inserted after `<af:showDetailItem ...>` and the page fragment in the editor will resemble Figure 14–7.

Example 14–3 Example Edit Task Flow Binding Code

```
<af:region value="#{bindings.EmpSummaryTF2.regionModel}"
 id="EmpSummaryTF2"/>
```

Figure 14–7 ADF Faces Components in Page Fragment Editor



Now that you have created the Main Area page fragment, you must wrap it in an ADF task flow.

6. In the Create ADF Task Flow dialog:

   - Enter a descriptive name for the task flow.

     For example, enter `def_main_task-flow-definition.xml`.

   - Make sure that the **Create as Bounded Task Flow** and the **Create With Page Fragments** boxes are checked.

     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 14–8.

Figure 14–8 Page Fragment itemNode

9. To load the menu metadata:

Note: The menu data accomplishes several important jobs for you:

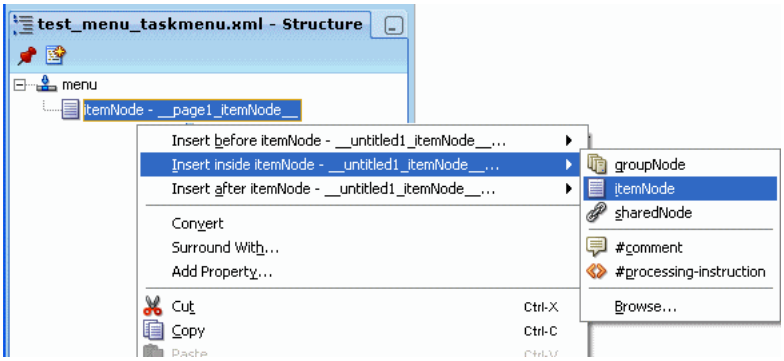
- It defines properties of the page for you. For instance, it will be displayed in no-tab mode or with dynamic tabs, and define the width of Regional Area.
- Can create a task list menu for each page.
- Can create labels for groups of tasks.

---

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 Section 14.2.1.1.1, "How to Create an Applications Menu."

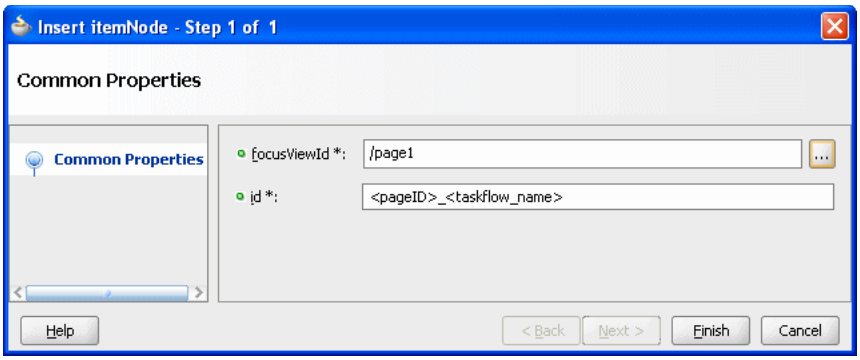
b. In the `test_menu_taskmenu.xml` structure view menu tree, shown in Figure 14–9, right-click the **itemNode** item and choose **Insert inside itemNode <*task_flow_name*> > itemNode**.

Figure 14–9 Task-Flow Item Node Menu Choices



The Insert itemNode - Common Properties dialog, shown in Figure 14–10, is displayed.

Figure 14–10 Insert itemNode - Common Properties Dialog



10. To the right of the **focusViewId** field, click the ellipsis to display the Edit Property dialog.

    In the dialog, choose the **ADFc View Activity id** of the page under which you are registering the task flow, then click **OK**.

11. Enter a unique ID using this standard format:

    <pageID>_<taskFlowName>

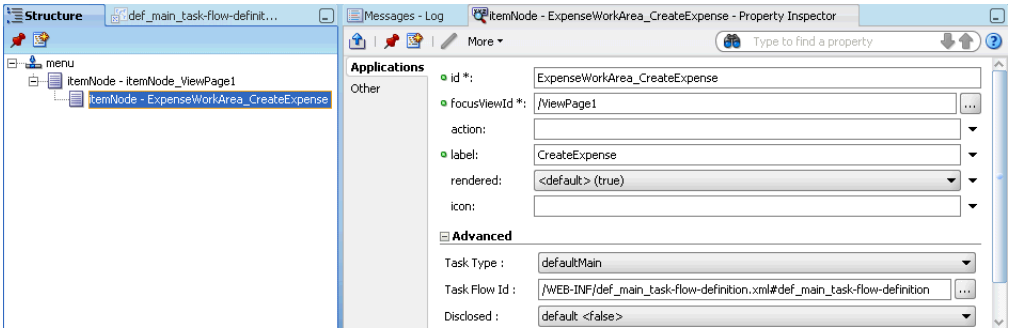
    This ID example consists of the concatenated page ID (or page name), an underscore, and the task flow name. For example, **ExpenseWorkArea_CreateExpense**.

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 14–11.

Figure 14–11 Task Flow Property Inspector



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 Section 14.2.3.4, "Supporting No-Tab Workareas"), 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:

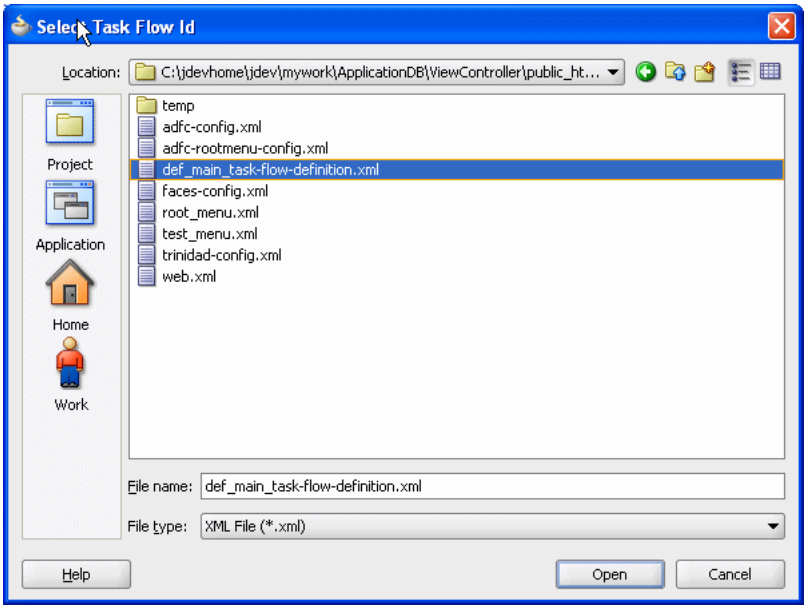
    - **Task Type**: **defaultMain** (task flow is displayed by default whenever the page is rendered).

      The Data Control Scope should have been set to `isolated`, inside the task flow definition for any taskflow 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 14–12, and browse to the task-flow definition location. By default, following the standard naming structure, the location will be in *`path_to_application directory`*`\ViewController\public_html\WEB-INF`.

Figure 14–12 Select Task Flow Id Dialog

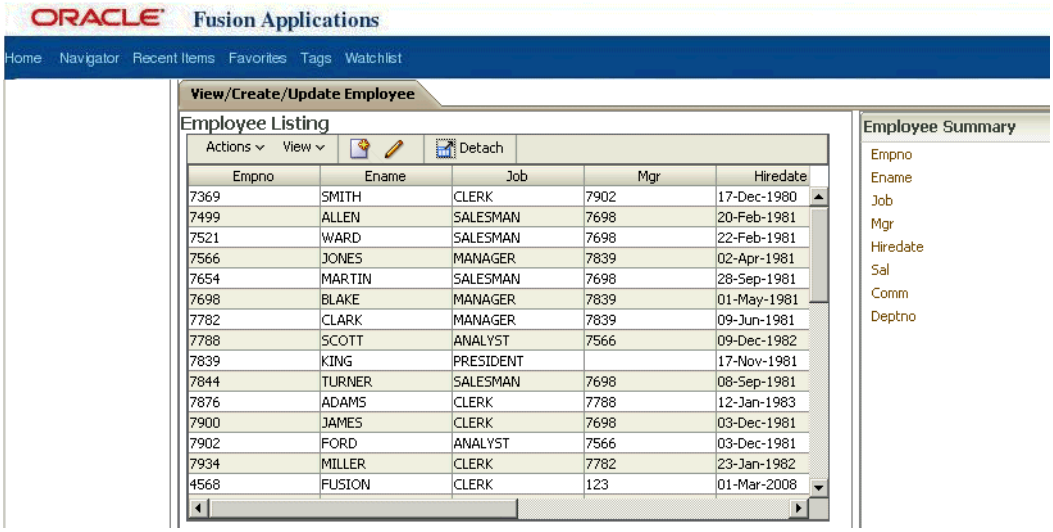


    Click **Open** to automatically enter the location in the **Task Flow** field. The task flow ID is a concatenation of the file location for the task-flow definition and the task-flow name. It typically resembles `/WEB-INF/MyTaskFlow.xml#MyTaskFlow`. The Property Inspector for the itemNode should resemble the example shown in Figure 14–11, "Task Flow Property Inspector".

16. To run the JSPX page, select the page in the Application Navigator, right-click the page file, and choose **Run**.

    The new page, shown in Figure 14–13, is displayed in a web browser.

Figure 14–13 Rendered Page in 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 and Regional task flows for application menus are shown in Table 14–2.

Table 14–2 itemNode Properties for Main and Regional Task Flows for Application Menus

itemNode Property	Property Value	What Happens on the Rendered Page
`taskType`	**Note:** `taskType` can have four values: ▪ `dynamicMain` ▪ `defaultMain` ▪ `defaultRegional` ▪ `taskCategory`	▪ If the value is `dynamicMain`, 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 Section 14.2.3.4, "Supporting No-Tab Workareas.") ▪ If the value is `defaultMain`, the page contains a tab already running this task in the main area. If the no-tabs model is used, only one itemNode should be defined as a `defaultMain`. (See Section 14.2.3.4, "Supporting No-Tab Workareas.") ▪ If the value is `defaultRegional`, the task is loaded into the regional area.
`label`	String	**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 Section 14.2.3.4, "Supporting No-Tab Workareas"), 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.
`taskFlowId`	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`	
`reuseInstance` (optional)	`True` or `False`	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 Section 14.2.3.4, "Supporting No-Tab Workareas.")

Table 14–2 (Cont.) itemNode Properties for Main and Regional Task Flows for Application Menus

itemNode Property	Property Value	What Happens on the Rendered Page
`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 DataControl methods: ▪ openMainTask ▪ discloseRegionalTask ▪ collapseRegionalTask ▪ navigate ▪ openSubTask In dynamic tabs mode, when looking for a match of a 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 only reuse a tab if you pass in a null keylist.
loadPopup	`True` or `False`	See Section 14.2.3.5, "Implementing the Task Popup." Provides a way to load the task flow into a Popup when the user clicks one of the Tasks List links.
`loadDependentFlow`	True or False	No-tab navigation mode can load a main flow and a dependent flow simultaneously, while displaying only one flow at a time. (See Section 14.2.3.4, "Supporting No-Tab Workareas.") The UI Shell is limited to 12 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 no-tab navigation model. Instead of having only one region for the no-tab navigation model, there are two regions: one for the main 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 flow is loaded, only the dependent flow region is shown, and the main flow region is hidden. When the dependent flow is closed, the main 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 flow will close both the existing main flow and, if any, the existing dependent flow. Loading a new task flow in the dependent flow will replace only the existing dependent flow, if any, and leave the main flow intact.
`forceRefresh`	True or False	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.

Table 14–2 (Cont.) itemNode Properties for Main and Regional Task Flows for Application Menus

itemNode Property	Property Value	What Happens on the Rendered Page
`stretch`	True (default) or False	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 render with a scroll bar, if necessary, in the Local Area. (This facet is contained within an af:panelGroupLayout with layout=scroll.)
`disclosed` (optional)	`True` or `False`	All the task flows will render 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.
`active`	`True` or `False`	Default is `false`. Task flow definitions use conditional activation. There are a number of 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 will need to change the `active` property on the itemNode to `True`.
`taskFlow`		The Tasks List is exposed as a task flow. See Section 14.2.4, "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 task type set to `dynamicMain`. The destination 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"/>`

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 Section 14.2.2, "How to Add Default Main Area Task Flows to a Page," to insert the appropriate **itemNode** properties listed inTable 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.

You need to 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 prior to the item node of tree structure and tree versions:

```
<itemNode id="__YourPage_itemNode__FndTasksList"
        focusViewId="/YourPage" label="#{applcoreBundle.TASKS}"
        taskType="defaultRegional"
```

Note: The taskFlowId value path *must* appear in a single line to avoid an exception during runtime.

```
        taskFlowId="/WEB-INF/oracle/apps/fnd/applcore/patterns/
                    uishell/ui/publicFlow/TasksList.xml#TasksList"
        disclosed="true"
        parametersList="fndPageParams=
                            #{pageFlowScope.fndPageParams}"/>
```

where:

- Id needs to 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.
- `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 shown above 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 Section 14.19, "Using the Single Object Context Workarea."

14.2.3.2 Grouping 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 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 Section 14.2.3.4, "Supporting No-Tab Workareas"), 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.

   To run the page, right-click the JSPX page file in the Projects tree view and choose **Run**.

   Check 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. It can 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

```
<itemNode id="__ServiceRequest_itemNode__toTestPage1"
                focusViewId="/ServiceRequest" label="Go to different page"
navigateViewId="/TestPage1"
                taskType="dynamicMain"
taskFlowId="/oracle/apps/fnd/applcore/patterns/demo/SRTree.xml#SRTree"/>
```

14.2.3.4 Supporting No-Tab Workareas

Product teams 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.

Example 14–5 Implementing No-Tab Workarea

```
<itemNode focusViewId="/SelTestWorkarea" id="stp1" taskType="dynamicMain"
  taskFlowId="/WEBINF/oracle/.../ProductMainFlow.xml#ProductMainFlow"
  label="#{adfBundle
      ['oracle.apps....SelTestWorkarea_taskmenuBundle'].DEFINE_PRODUCT}"
  isDynamicTabNavigation="false"/>
```

Note that the default value of `isDynamicTabNavigation` is true.

You also can set no-tab mode declaratively in the Property Inspector:

1. Select the itemNode from the Structure window.
2. Go to Property Inspector.
3. Select **Advanced > Page > Dynamic Tab Navigation**. Selecting the Page tab lets you set attribute values for Page-level item nodes.
4. Set the Dynamic Tab Navigation property to **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 product teams to:

- Load their 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, 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 again contain a dynamic region defined in it. On user click, the UI Shell will load the product team's task flow into the dynamic region, and show the modal `af:popup` panelWindow without any buttons. Therefore, the 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. The dialog title will be set according to the label mentioned in the menu meta data 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 launched.

Developer Implementation

There are several considerations to keep in mind when you implement the Task Popup:

- For a dynamicMain Task item 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 launches the task flow.

Example 14–6 Example Use of loadPopup Property

```
<itemNode id="__ServiceRequest_itemNode__ChooseSR"
          focusViewId="/ServiceRequest" label="Choose SR"
          taskType="dynamicMain" taskFlowId="/WEB-INF/ChooseSR.xml#ChooseSR"
          parametersMap="#{pageFlowScope.Mybean.Map}"
          loadPopup="true"/>
```

- Developers 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`.
- Teams cannot have a UIShellMainArea page template or any other templates inside the popup task flow.
- Developers 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 Section 16.4.1.3, "Implementing OK and Cancel Buttons in a Popup."
- Create another method for the **OK** button that calls the method in Example 16–5, 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 Section 16.4.1.3, "Implementing OK and Cancel Buttons in a Popup."
- Developers then can pass the parameters directly from the managed bean to the openMainTask API bindings for the popup task flow page to launch 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.

14.2.4 How to Pass Parameters into Task Flows from Tasks List

itemNodes with **taskType** of **dynamicMain**, **defaultMain**, and **defaultRegional** have parameter support. In addition to specifying the **taskFlowId** to load when the user clicks a Task link, developers 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.

Example 14–7 Using the parametersList Property

```
<itemNode id="__ServiceRequest_itemNode__SRDefault" focusViewId="/ServiceRequest"
label="Pending Service Requests"
taskType="dynamicMain"
taskFlowId="/oracle/apps/fnd/applcore/patterns/demo/SRTable.xml#SRTable"
parametersList="param1=value1;param2=value2;param3=#{ELForValue3}"/>
```

methodParameters can be used for passing a Java object into the task flow that is specified in the taskFlowId parameter. Use the setCustomObject() API in FndMethodParameters for setting the Java object.

Example of Passing a Java Object Using openMainTask

Bind the methodParameters parameter value to a managed bean property. Example 14–8 shows the method action binding in the page definition of the page fragment that calls openMainTask. Also see Table 14–2, " itemNode Properties for Main and Regional Task Flows for Application Menus".

Example 14–8 Method Action Binding to Call openMainTask

```
    <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" 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" NDType="java.lang.Boolean"/>
      <NamedData NDName="forceRefresh" NDType="java.lang.Boolean"/>
      <NamedData NDName="loadDependentFlow" NDValue=""
                 NDType="java.lang.Boolean"/>
      <NamedData NDName="methodParameters"
NDValue="#{TestOpenMainTaskMBean.fndMethodParams}"

NDType="oracle.apps.fnd.applcore.patterns.uishell.ui.bean.FndMethodParameters"/>
    </methodAction>
```

Code in the managed bean for passing a hashmap to the task flow would resemble Example 14–9.

Example 14–9 Example Code for Passing a Hashmap

```
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;
  }
```

Then, in the managed bean of the task flow, the Java object can be read, as shown in Example 14–10.

Example 14–10 Reading Java Object in Managed Bean

```
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;
  }
```

where testValue is bound to an `af:outputText` value attribute, such as `<af:outputText value="#{TestPanelSplitter1MBean.testValue}"/>`, in the page fragment of the task flow.

14.2.5 How to Launch 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 launch 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. Once 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. Once the URI for the file is generated, this is set on the URLView activity and an action expression is set on the task link to launch the URL view. UI Shell also needs to 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.

Limitations

Product teams can only launch data files that are part of their webApp, such as `/oracle/apps/fin/acc/file1.xls`. This feature only supports the launch of data files through the task list. Any other URI paths, such as a JSPX or a JSF page, are not supported.

Developer Implementation

For a `dynamicMain` task item that you would like to use to launch 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 14–11.

Example 14–11 Specifying the File URI Path

```
<itemNode id="__ServiceRequest_itemNode__ChooseSR"
          focusViewId="/ServiceRequest" label="Download File"
          taskType="dynamicMain" filePath="/WEB-INF/oracle/apps/Accounts.xls" />
```

You can specify any file type, such as xls, doc, pdf, txt, rtf, and ppt, that is within the application.

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 only run for a user 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" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Application menus and taskList 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 these 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 of these conditions are 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 anything. 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 or not the customer is a non-profit 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" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

If your UI Shell pages are secured by ADF Security, you must add a policy, similar to Example 14–12, to the jazn-data.xml file and the system-jazn-data.xml file.

Example 14–12 Adding a Security Policy to the jazn-data.xml File

```
<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>
         <name>oracle.apps.fnd.applcore.uicomponents.view.pageDefs.oracle_apps_
fnd_applcore_templates_UIShellPageDef</name>
         <actions>view</actions>
     </permission>
     <permission>
        <class>oracle.adf.controller.security.TaskFlowPermission</class>

<name>/oracle/apps/fnd/applcore/patterns/uishell/MainArea.xml#MainArea</name>
        <actions>view</actions>
     </permission>
     <permission>
        <class>oracle.adf.controller.security.TaskFlowPermission</class>

<name>/oracle/apps/fnd/applcore/patterns/uishell/RegionalArea.xml#RegionalArea</na
me>
        <actions>view</actions>
     </permission>
     <permission>
        <class>oracle.adf.controller.security.TaskFlowPermission</class>

<name>/WEB-INF/oracle/apps/fnd/applcore/patterns/uishell/ui/publicFlow/TasksList.x
ml#TasksList</name>
        <actions>view</actions>
     </permission>
  </permissions>
</grant>
```

Task Flow Example

Bounded task flows are secure by default, and require the policy shown in Example 14–13.

Example 14–13 Required Policy for Bounded Task Flows

```
<permission>  <class>oracle.adf.controller.security.TaskFlowPermission</class>
<name>/WEB-INF/audit-expense-report.xml#audit-expense-report</name>
<actions>view</actions></permission>
```

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 pre-check of the same permission? That's where an Expression Language expression comes in.

Example 14–14 shows the generic Expression Language expression being used to perform a pre-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 Generic Expression Language Expression Used for Task Flow Permission Pre-check

```
rendered =
"#{securityContext.userGrantedPermission['permissionClass=oracle.adf.controller.se
curity.TaskFlowPermission;
               target=/WEB-INF/audit-expense-report.xml#audit-expense-report;
               action=view']}"
```

Example 14–15 shows the task flow-specific Expression Language.

Example 14–15 Task Flow-specific Expression Language Expression Used for Task Flow Permission Pre-check

```
rendered="#{securityContext.taskflowViewable[/WEB-INF/audit-expense-report.xml#aud
it-expense-report]}"
```

Note that both of these checks actually go directly against the policy store; that is, they don't interrogate the task flow definition. This avoids the overhead of loading a large number of ADF artifacts to render links and menus.

14.4 Controlling the State of Main and Regional Area Task Flows

UI Shell tasks to open up or close a Main Area tab are exposed as data control methods so that you easily 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.

14.4.1 How to Control Main Area Task Flows

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 Section 14.2.3.4, "Supporting No-Tab Workareas"), 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.

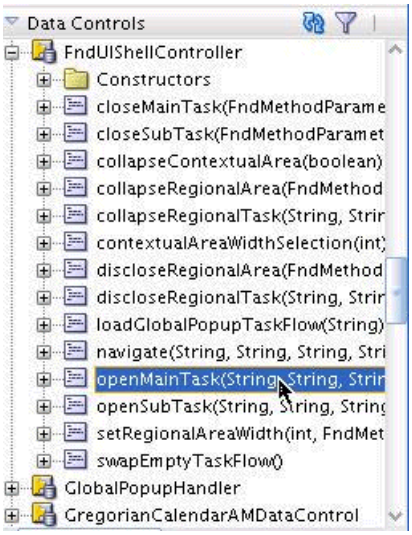
- FndUIShellController.closeMainTask. See Section 14.4.1.1, "closeMainTask History" for more information.

For example, to open or close a Main Area tab, drag and drop the appropriate data control method to create the UI affordance. Having specified the parameter values into these methods, user clicks will prompt 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 14–14.

Figure 14–14 Selecting openMainTask from Data Controls



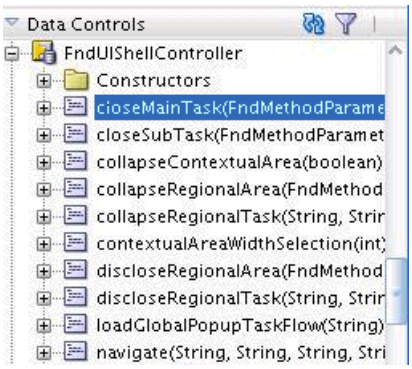
2. Drag openMainTask and drop it onto the page fragment. When you do, the Applications Context menu shown in Figure 14–15 displays so you can choose one of the three options.

Figure 14–15 Selecting an Open Option from the Applications Context Menu

To use the closeMainTask data control method:

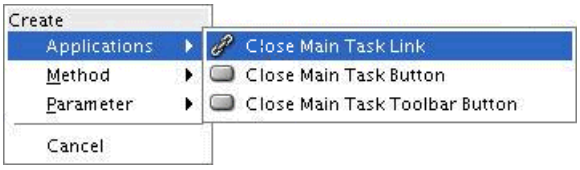
1. Expand the Data Controls and select the closeMainTask item, as shown in Figure 14–16.

Figure 14–16 Selecting closeMainTask from Data Controls



2. Drag closeMainTask and drop it onto the page fragment. When you do, the Applications Context menu shown in Figure 14–17 displays so you can choose one of the three options.

Figure 14–17 Selecting a Close Option from the Applications Context Menu



Two APIs, shown in Example 14–16, are exposed to open and close a Main Area tab.

Example 14–16 APIs Exposed to Open and Close a Main Area Tab

```
/**
   * Opens a Main Area task.
   *
   * @param taskFlowId Task flow to open
   * @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 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 reuseInstance Default true. If true, refocus an existing instance
   *                      of the task flow, if such a one exists, without
   *                      opening a new instance of the task flow. If false,
   *                      always open a new instance of 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 reinitializaiton and the absence of
   * parameter values does not. forceRefresh true always causes reinitialization
   * of the task flow regardless of the value for parametersList.
   * @param loadDependentFlow Effective only in No-Tab navigation model.
   *                          Defaults to false. When set to true, the specified
   *                          task flow is loaded into the dependent region
   *                          of the No-Tab navigation model, preserving the
   *                          state of the main flow.
```

```
   * @param methodParameters From Drop 6 Build 7, this can be used for passing
   *                         java object into the task flow that's specified
   *                         in taskFlowId parameter. Use setCustomObject() API
   *                         in FndMethodParameters for setting the java object.
   * @return For internal Contextual Event processing
   */
  public FndMethodParameters openMainTask(String taskFlowId,
                                          String keyList,
                                          String parametersList,
                                          String label,
                                          Boolean reuseInstance,
                                          Boolean forceRefresh,
                                          Boolean loadDependentFlow,
                                          FndMethodParameters methodParameters)

/**
   * Closes the current focused task flow.
   *
   * @param methodParameters For future implementation. No-op for now.
   * @return For internal Contextual Event processing
   */
  public FndMethodParameters closeMainTask(FndMethodParameters methodParameters)
```

Bind the `methodParameters` parameter value to a managed bean property. Example 14–17 shows the method action binding in the page definition of the page fragment that calls `openMainTask`.

Example 14–17 Method Action Binding in Page Definition of Page Fragment That Calls openMainTask

```
    <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" 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" NDType="java.lang.Boolean"/>
      <NamedData NDName="forceRefresh" NDType="java.lang.Boolean"/>
      <NamedData NDName="loadDependentFlow" NDValue=""
                 NDType="java.lang.Boolean"/>
      <NamedData NDName="methodParameters"
NDValue="#{TestOpenMainTaskMBean.fndMethodParams}"

NDType="oracle.apps.fnd.applcore.patterns.uishell.ui.bean.FndMethodParameters"/>
    </methodAction>
```

Example 14–18 shows code in a managed bean for passing a hashmap to the task flow.

Example 14–18 Sample Code in a Managed Bean for Passing a Hashmap to the Task Flow

```
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;
}
```

Then, in the managed bean of the task flow, the Java object can be read, as shown in Example 14–19.

Example 14–19 Reading Java Object in a Managed Bean

```
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;
  }
```

Where testValue is bound to an af:outputText value attribute, such as `<af:outputText value="#{TestPanelSplitter1MBean.testValue}"/>`, in the page fragment of the task flow.

14.4.1.1 closeMainTask History

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.

`MainAreaHandler.handleOpenMainTaskEvent` 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 launched, the current tab is set as the previous tab for the managed bean for the new tab.

A disclosure listener, `MainAreaBackingBean.setLastDisclosedItem`, handles user clicks in the tab UI. When the user clicks a tab, two events fire: 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 end up in 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`. When a new task flow is opened, the task flow ID and its associated parameter values are pushed onto 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 Section 14.2.3.4, "Supporting No-Tab Workareas."

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 panelAccordian.

Declarative support: (to allow the developer to specify the initial state of the following on loading a Work Area JSPX page)

- Within the Regional Area, whether or not a Regional Area Task Panel is collapsed or disclosed.
- A given Regional Area panel that is disclosed on initial rendering of the page should honor its assigned pixel height to determine how much screen real estate it occupies.

Programmatic support: (to allow the developer 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, the developer 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 gesture 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.

Developer Implementation

- 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 14–20.

Example 14–20 Sample Menu File Entry

```
<itemNode id="itemNode_SvcCenter"
           label="#{adfBundle['oracle.apps.fnd.applcore.patterns.demo.patterns_
demo_menuBundle'].SERVICE_CENTER}"
           action="adfMenu_SvcCenter" focusViewId="/SvcCenter"
           isDynamicTabNavigation="false" regionalAreaWidth="250"
isRegionalAreaCollapsed="false">
```

- If these properties are not set in the menu for the top-level item node that represents the page, the default values used are:

  ```
  regionalAreaWidth="256"
  isRegionalAreaCollapsed ="false"
  ```

- For programmatic control, drag and drop the corresponding method from the **FndUIShellController** data control.
  - discloseRegionalArea
  - collapseRegionalArea
  - setRegionalAreaWidth

Two APIs, shown in Example 14–21, are exposed as data control methods under FndUIShellController.

Example 14–21 APIs Exposed as Data Control Methods Under FndUIShellController

```
/**
   * 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 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 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 reinitializaiton and the absence of
   * parameter values does not. forceRefresh true always causes reinitialization
   * of the task flow regardless of the value for parametersList.
   * @param methodParameters For future implementation. No-op for now.
   * @return For internal Contextual Event processing
   */
  public FndMethodParameters discloseRegionalTask(String taskFlowId,
                                                  String keyList,
                                                  String parametersList,
                                                  String label,
                                                  Boolean forceRefresh,
                                                  FndMethodParameters
methodParameters)
```

```
/**
 * 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 key-value pairs. For example,
 *                "key1=value1;key2=value2".
 * @param methodParameters For future implementation. No-op for now.
 * @return For internal Contextual Event processing
 */
public FndMethodParameters collapseRegionalTask(String taskFlowId,
                                        String keyList,
                                        FndMethodParameters methodParameters)
```

Limitations

- Declarative support allows the `inflexibleHeight` property to control the pixel height of the Regional panel. Programmatic support does not have this allowance.
- Programmatic support allows for `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 Task without disclosing the task is not supported.
- Multiple Regional 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.

14.4.3 How to Control the State of the Contextual Area Splitter

This section discusses the declarative and programmatic means of controlling the state of the Contextual Area splitter.

The UI Shell must expose the means to control the disclosure state of the Contextual Area.

Declarative support lets the developer specify the initial state when loading a Work Area JSPX page. It determines whether or not the Contextual Area (as a whole) is collapsed or disclosed.

Programmatic support lets the developer 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, the developer 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 gesture by the user, such as a button click or menu selection.

Samples of Expected Behavior

- A Work Area page (JSPX) loads with the Contextual Area collapsed or disclosed when the page renders, based on the declarative setting. If the Work Area is loaded as a result of a Main Menu invocation, declarative options always are used for the disclosure state.

- If a Work Area loads as a result of a page navigation from another Work Area, programmatically set options may override declarative settings.

14.4.3.1 Implementing the Contextual Area Splitter

- Extend the contextual-area-task-flow-template Task Flow Template into the page task flow, as shown in Example 14–22.

Example 14–22 Extending the Task Flow Template

```
<template-reference>

<document>/oracle/apps/fnd/applcore/patterns/uishell/templates/contextual-area-tas
k-flow-template.xml</document>
   <id>contextual-area-task-flow-template</id>
</template-reference>
```

- Specify values for the contextual area splitter position and the collapsed state in the menu for the item node that represents the page using `contextualAreaWidth` and `contextualAreaCollapsed` properties. A sample entry in the menu file will resemble Example 14–23.

Example 14–23 Example of contextualAreaWidth and contextualAreaCollapsed Properties

```
<itemNode focusViewId="<focus_view_id>" id="<page_id>" label="<page_label>"
     taskType="dynamicMain"

taskFlowId="/WEB-INF/page2-task-flow-definition.xml#page2-task-flow-definition"
     contextualAreaCollapsed="true"
     contextualAreaWidth="0"/>
```

- If these properties are not set in the menu for the top-level item node that represents the page, these default values are used:

  ```
  contextualAreaWidth="256"
  contextualAreaCollapsed ="false"
  ```

- For programmatic control, drag and drop the corresponding method from the data control named FndUIShellController:
  - `collapseContextualArea`
  - `contextualAreaWidthSelection`
- To set these values when opening a new task, drag and drop the `openMainTask` method from FndUIShellController and pass in the `contextualAreaWidth` and `contextualAreaCollapsed` parameters through "methodsParameters > NamedData" as shown in Example 14–24.

  Set the method in the page managed bean to set the `contextualAreaWidth` and `contextualAreaCollapsed` values, as shown in Example 14–24.

Example 14–24 Setting the contextualAreaWidth and contextualAreaCollapsed Values for openMainTask

```
<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/page6-task-flow-definition.xml#
                              page6-task-flow-definition"
            NDType="java.lang.String"/>
      <NamedData NDName="keyList" NDValue="" NDType="java.lang.String"/>
      <NamedData NDName="parametersList" NDType="java.lang.String"/>
      <NamedData NDName="label" NDType="java.lang.String"/>
      <NamedData NDName="reuseInstance" NDType="java.lang.Boolean"/>
      <NamedData NDName="forceRefresh" NDType="java.lang.Boolean"/>
      <NamedData NDName="loadDependentFlow" NDType="java.lang.Boolean"/>
      <NamedData NDName="methodParameters"
            NDValue="#{<ManagedBean.Method>}"
            NDType="oracle.apps.fnd.applcore.patterns.uishell.ui.
                                    bean.FndMethodParameters"/>
```

- For setting 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 14–25.

  Set the method in the page managed bean to set the `contextualAreaWidth` and `contextualAreaCollapsed` values, as shown in Example 14–25.

Example 14–25 Setting the contextualAreaWidth and contextualAreaCollapsed Values for navigate

```
<methodAction id="navigate" RequiresUpdateModel="true" Action="invokeMethod"
       MethodName="navigate" IsViewObjectMethod="false"
       DataControl="FndUIShellController"
       InstanceName="FndUIShellController.dataProvider"
       ReturnName="FndUIShellController.methodResults.navigate_
FndUIShellController_dataProvider_navigate_result">
    <NamedData NDName="viewId" NDType="java.lang.String"/>
    <NamedData NDName="webApp" NDType="java.lang.String"/>
    <NamedData NDName="pageParametersList" NDType="java.lang.String"/>
    <NamedData NDName="navTaskFlowId" NDType="java.lang.String"/>
    <NamedData NDName="navTaskKeyList" NDType="java.lang.String"/>
    <NamedData NDName="navTaskParametersList" NDType="java.lang.String"/>
    <NamedData NDName="navTaskLabel" NDType="java.lang.String"/>
    <NamedData NDName="methodParameters"
        NDValue="#{<ManagedBean.Method>}"
        NDType="oracle.apps.fnd.applcore.patterns.uishell.ui.
                                    bean.FndMethodParameters"/>
</methodAction>
```

14.4.4 Sizing Regional Area Panels

Multiple Regional Area panels will 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 for each panel, developers can set the pixel height for each of the panels by specifying the `inflexibleHeight` property in the itemNode that represents a Regional Area panel, as shown in Example 14–26.

Example 14–26 Using inflexibleHeight to Set Panel Height

```
<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 Menu (also known as the main menu), and the Preferences Menu.

The Navigator Menu 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 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

Example of Global Menus

The global menu model presents a *cascading* appearance, shown in Example 14–31.

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 (**...**) display. The entire entry will display in a tool tip when the pointer hovers over the entry.
- Parent and children will not be split in 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 J2EE applications.

- **Navigator Menu**: This is the global menu that displays 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 tasklist to all other preference 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, Table 14–4, and Table 14–15 list the menu attributes added by Applications Core to the menu XML above what is provided by Oracle ADF.

Table 14–3 <groupNode> Attributes

Attribute	Data Type	Required	Description
`labelKey`	xsd:string	N	Bundle key used for label; the key will be looked up in the resource bundle specified by the resourceBundle attribute of <menu>.

Table 14–4 <itemNode> Attributes for Global Menus

Attribute	Data Type	Required	Description
`webApp`	xsd:string	Y	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.
`focusViewId`	xsd:string	N	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.
`securedResourceName`	xsd:string	N	The resource name that is used for securing the item node.
`applicationStripe`	string		(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.
`parametersList`	string		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, "param1=value1;param2=value2."
`pageParametersList`	string		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, "param1=value1;param2=value2". If the Expression Language expression evaluates to an Object, the toString value of that Object will be passed as the value of the parameter.
`destination`	string		The `destination` attribute is supported on the item nodes for the Navigator menu. The `destination` should only be used 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 Menu

The Navigator menu, shown in Figure 14–18, is rendered when the Navigator link is clicked on the UI Shell.

Figure 14–18 Navigator Menu Example

14.5.1.3 Implementing a Global Menu

> **Note:** The Navigator menu 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 menu metadata may be pointing to target workarea 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 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` 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 14–27.

Example 14–27 Sample JAAS Filter

```
<filter>
        <filter-name>JpsFilter</filter-name>
        <filter-class>oracle.security.jps.ee.http.JpsFilter</filter-class>
        <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>
          <param-value>crm</param-value>
        </init-param>
<init-param>
          <param-name>oracle.security.jps.jaas.mode</param-name>
          <param-value>subjectOnly</param-value>
        </init-param>
</filter>
```

application.name, as shown in the example, in `web.xml` is the application family value. The choices are crm, fscm, and hcm. This value is used to create the stripe in LDAP.

2. Update `weblogic-application.xml`. As shown in Example 14–28, set the application-param that has the param-name **jps.policystore.migration** to OFF.

Example 14–28 Setting jps.policystore.migration to OFF

```
<application-param>
    <param-name>jps.policystore.migration</param-name>
    <param-value>OFF</param-value>
</application-param>
```

3. In `weblogic-application.xml`, make sure the application-param that has the param-name **jps.policystore.applicationid** is set to the correct stripe, as shown in Example 14–29. This is the same as the application.name property of `web.xml`.

Example 14–29 Setting jps.policystore.applicationid to the Correct Stripe

```
<application-param>
    <param-name>jps.policystore.applicationid</param-name>
    <param-value>crm</param-value>
</application-param>
```

4. Add the following entry in **web.xml**:

```
<listener>
<listener-class>oracle.apps.fnd.applcore.menu.service.MenuFragmentServiceContex
tListener</listener-class>
</listener>
```

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, launch WebLogic Server *after* setting the JAVA_OPTIONS environmental 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, you should already have defined roles, principals, and actions in the database.

Functional security will always prevent a user from accessing a page or task flow that the user does not have access to. 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, but not to show an entry under the Employee Self-Service category that also led to the same page. See Example 14–30.

Example 14–30 Expression Language Expression to Evaluate a User's Access Rights

```
rendered="#{securityContext.userInRole['EMPLOYEE_ROLE']}"
```

The Expression Language expression should never check the pageDef. However, you can use the Expression Language expression to check security of a person's role since that is in LDAP.

- The `applicationStripe` attribute determines which LDAP stripe is checked for the `securedResourceName`.

14.5.3 How to Create the Navigator Menu

Menu files will be references through MDS. This means they can be located in a table or in a file system directory. Determine where this directory will be now. This is where your `root_menu.xml` and other menu files will be located. For Global Menu attributes, see Section 14.5.1.1, "Menu Attributes Added by Oracle Fusion Middleware Extensions for Applications (Applications Core)."

1. Create the root menu.

   Example 14–31 shows a sample root menu.

Example 14–31 Example of a Navigator Menu

```
<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}">
      <itemNode id="itemNode_my_information_my_portrait"
         label="#{bundleVar.MY_PORTRAIT}" focusViewId="/MyPortrait"
webApp="HcmCore"

securedResourceName="oracle.apps.hcm.people.portrait.ui.page.MyPortraitPageDef"
    <groupNode id="groupNode_my_information_compensation"
      idref="_groupNode_my_information_compensation_"
      label="#{menuBundle['oracle.apps.menu.ResourcesAttrBundle'].COMPENSATION}">
      applicationStripe="hcm"
    </groupNode>
    <groupNode id="groupNode_my_information_career"
      idref="_groupNode_my_information_career_"
      label="#{menuBundle['oracle.apps.menu.ResourcesAttrBundle'].CAREER}">
         <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.ManageGoalsWorkAreaP
ageDef"/>
              <itemNode id="itemNode_my_information_performance_management"

label="#{menuBundle['oracle.apps.menu.ResourcesAttrBundle'].PERFORMANCE}"
        applicationStripe="hcm"
                 focusViewId="/PerformanceWorkArea" webApp="HcmTalent"
```

```
securedResourceName="oracle.apps.hcm.performance.documents.publicUi.page.Performan
ceWorkAreaPageDef"/>
    </groupNode>
    <groupNode id="groupNode_my_information_procurement"
      idref="_groupNode_my_information_procurement_"
      label="#{menuBundle['oracle.apps.menu.ResourcesAttrBundle'].PROCUREMENT}">
         <itemNode id="itemNode_my_information_purchase_requisitions"
            label="#{menuBundle['oracle.apps.menu.ResourcesAttrBundle'].PURCHASE_
REQUISITIONS}"
            focusViewId="/PrcPorCreateReqWorkarea" webApp="Procurement"

securedResourceName="oracle.apps.prc.por.createReq.publicUi.page.PrcPorCreateReqWo
rkareaPageDef" applicationStripe="hcm"/>
               <itemNode id="itemNode_my_information_self_service_receipts"

label="#{menuBundle['oracle.apps.menu.ResourcesAttrBundle'].RECEIPTS}"
                  applicationStripe="hcm"
                  focusViewId="/RcvSelfServWorkarea" webApp="Logistics"

securedResourceName="oracle.apps.scm.receiving.selfService.workarea.ui.page.RcvSel
fServWorkareaPageDef"/>
       applicationStripe="hcm"/>
    </groupNode>
  </groupNode>
</menu>
```

2. Create the application's Navigator menu files.

   The next files in the menu hierarchy can contain groupNodes that appear as non-clickable categories, itemNodes that are clickable to launch a page, or references to more menu files. If itemNodes were included that were not deployed, they will not appear since a check is done against the deployment tables of what was deployed. Applications Core requires that if a groupNode has no children, which could happen through security enforcement, the groupNode itself will not be rendered.

14.5.3.1 Rendering the Navigator Menu as Pull-down Buttons

There are situations, particularly with more simple applications, when the default enterprise-level menu structure is not suitable. In these cases, you may want to display the Navigator menu as a series of pull-down buttons.

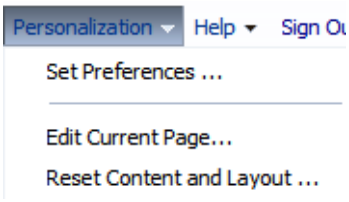
To switch the UI Shell rendering so the Navigator menu renders as pull-down buttons in a horizontal row, set the `isSelfService` attribute to "true" on the .jspx page that extends the UI Shell template. That is, inside the `<af:pageTemplate>` tag, add the following:

```
<f:attribute name="isSelfService" value="true"/>
```

14.6 Using the Personalization Menu

The Personalization menu options, shown in Figure 14–19, let you set your preferences, edit the current page, and reset the content and layout. The menu is supplied automatically by the UI Shell and requires no developer work.

Figure 14–19 Personalization Menu



The Preference Menu only appears if you have the ApplSession filter and mapping set up. See Section 47.2, "Configuring Your Project to Use Application User Sessions.".

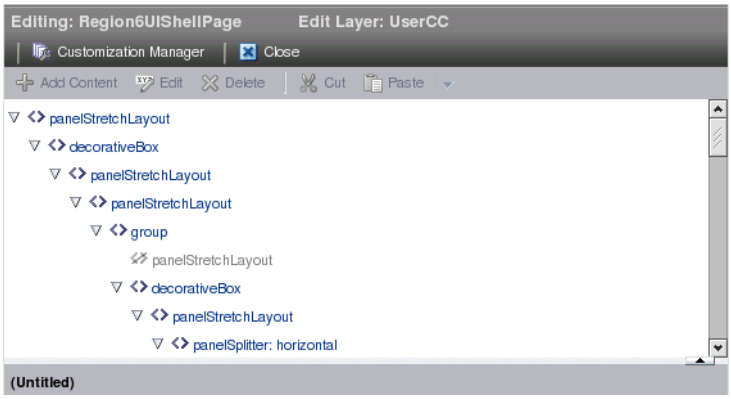
Set Preferences

The actual Preferences dialog, such as shown in Figure 14–22, is created by developers. See Section 14.7 for the details of how to implement the menu.

Edit Current Page

This option displays 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 14–20. Click **Close** to return to the page. Click **Customization Manager** to change the displayed page in Composer. For more information about the Customization Manager, see the "Customization Manager" section in the "Extending Runtime Editing Capabilities Using Oracle Composer" chapter, and the "Manage Customizations" section of the "Introduction to Oracle Composer" chapter of the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*. Note that changes are for just this user and therefore are called personalization. See Chapter 61, "Creating Customizable Applications".

Figure 14–20 Edit Current Page Display



Reset 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 taskflows are personalized on that page via Composer, they are not reset by this menu item.

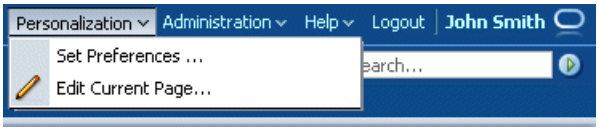
14.7 Implementing End User Preferences

Set Preferences, shown in Figure 14–21, is a link in the global area for 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 Section 14.5, "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 application-specific 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 can be stored in LDAP.

Figure 14–21 Preferences Menu Example



14.7.1 How to Use Preferences Link Navigation

The Preferences link from the global area will launch a Workarea that shows the preferences related to the currently-displayed page.

Links in the left hand side will allow navigation to any Preferences Workarea page within the entire Oracle Fusion product. This menu will be rendered using Applications Core menu federation abilities. Development teams will own the menu files.

If an application is not installed, or the user does not have access, the entry in the tasklist menu should not appear. If a user does not have access to a particular setting within a page, application teams need to use the `rendered` property with a security expression behind it.

For each application, there should be a preferences page. The preferences page will be found by looking for a page using the same path of the current application, but with 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 to 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.) Once in the preferences workarea, other links are available from the tasklist to more specific pages or task flows. (Links in the tasklist can contain other `focusViewIDs` that belong to the same application as the default preference page.)

When the first application is deployed, it should 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 only be navigated to from the task list. Therefore, from the Preferences link, the user always displays the more specific preferences page of that application.

14.7.2 How to Use the Preferences Workarea Page

A Preferences page will be like any other Workarea page. Preference values are not supported in integrated WLS LDAP, only an external LDAP is supported. The Tasks list will be loaded as a defaultRegional flow and the main area will be a defaultMain flow.

Workarea Title

Each Preferences page should display a title similar to {Category_name:Page_name}. This can be done though Expression Language and will not be created automatically from the framework.

The name that appears in the tasklist can be different from the page title. This is allowed since the tasklist name is generated from the tasklist preference distributed menu metadata, while the page title will be from the local page level menu metadata.

Tasklist / Navigation Pane

Each page needs its Application Menu metadata to specify that it wants the Preferences tasklist menu in the defaultRegional area as well as the defaultMain flow.

This menu can be a two-level menu having categories with links under each category.

Tasklist Federation

The tasklist will be a task flow that will contain links to all Preference pages throughout Oracle Fusion Middleware.

Each application will provide the preference menu files that contain tasklist links to preference pages delivered by that application. The preferences tasklist should follow the Navigator Menu 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 tasklist that renders will contain all the entries from all applications (filtered by security).

Individual menu files will be versioned 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.

Tasklist Only Can Link to Full Pages (not specific task flows)

The tasklist will not launch task flows dynamically, but will load a Preferences workarea page. This is because the tasklist menu needs to 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. Teams could use tabs if all flows are defaultMain if desired. See also Section 14.2.3.4, "Supporting No-Tab Workareas."

Tasklist Security

The tasklist 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.

Preference Settings

Settings will be a view activity in a task flow. It will follow other UX standards so it should be built using an Applications Panel. This means the action buttons will appear at the top.

Different Preference pages can change the same back end setting. This is up to Applications design. If this is needed, it should be stored in a common area, such as LDAP, or be in the General Preferences page.

14.7.3 How to Deploy Application Preferences Pages

Application preferences pages are deployed with the corresponding product pages.

14.7.4 How to Design General Preferences Content

The design should be similar to that shown in Figure 14–22.

Figure 14–22 General Preferences Example

14.7.5 How to Configure End-User Preferences

This section discusses how to set up and configure Preferences.

14.7.5.1 Implementing User General Preferences Flow

Once the WebLogic Server console is configured, 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` to **false** for the PreferencesUI page entry in the menu.
3. Add the following task flow as default regional under the preferences page entry: `"/WEB-INF/oracle/apps/fnd/applcore/pref/ui/mainflow/GeneralPreferencesFlow.xml#GeneralPreferencesFlow"`
4. The final menu entries for the page will appear similar to those shown in Example 14–32.

Example 14–32 Sample General Preferences Menu Entries

```
<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#Gen
eralPreferencesFlow"/>
</itemNode>
```

This should display the basic Preferences in the default regional area that can be launched to display sub-flows for each preference sub task (for instance, Accessibility and Appearance).

14.7.5.2 Using the Preferences Menu Model

The General Preferences 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. The Preferences menu will be maintained by the development team.

On any UI Shell page, the Global Area contains a Personalization menu that contains a Set Preferences link. This Preference link will redirect the user to a webApp-specific Preference page, depending upon the entry in Menu data.

Example 14–33 shows sample preferences menu data.

Example 14–33 Example Preferences Menu Data

```
<?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#Acceptanc
eFlow"
              />
  </itemNode>
</menu>
```

In the example menu XML file, each parent item node represents a Preferences UIShell page. Its child nodes refer to the webApp-specific flow in which the Preference page exists.

For example, the first itemNode refers to the preferencesA page that is part of the FND webApp. The ServiceFlow child node is a task flow that belongs to 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 preference 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 Workarea UI Shell pages in 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 preference page itemNode will display in the General Preferences Flow as navigate links. Therefore, all preference pages will have access to these flows.

14.7.5.3 Configuring User Session and ADF Security

To test the general preferences flows, you need to configure user session and ADF Security for the test application. See Chapter 47, "Implementing Application User Sessions."

When configuring ADF Security, there is no need to define users, because you already are using an OID store that will authenticate the users existing in the store.

14.7.5.4 Retrieving Preference Values

See Chapter 20, "Working with Localization Formatting."

14.7.5.4.1 How to Check Accessibility Mode by Using an Expression Language Expression A use case exists where the UI needs to 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 "How to Configure Accessibility Support in trinidad-config.xml" in "Developing Accessible ADF Faces Pages" in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

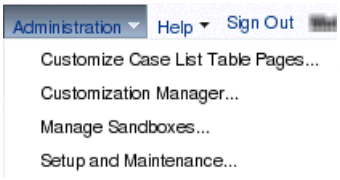
14.7.5.5 Implementing the Password Management Page

The Password link on the General Preferences page will point to the Password Management page from the Oracle Identity Management administration application. This page is maintained by the OIM team. For the Password link to redirect to the `pwdmgmt.jspx` page, the deployment information of the current application and the OIM administration application must be populated correctly in the ASK tables.

14.8 Using the Administration Menu

The Administration Menu, shown in Figure 14–23, is displayed only if the logged-in user has the appropriate privileges. See Section 14.8.1, "How to Secure the Administration Menu". The menu is supplied automatically by the UI Shell and requires no developer work.

Figure 14–23 Administration Menu



Customize Case List Table Pages ...

Select this option to customize the current page for multiple users using the customization layer picker dialog.

For information about customization, see Chapter 61, "Creating Customizable Applications" and the "Customizing Applications with MDS" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Customization Manager ...

Select this option to launch the Customization Manager.

For information about customization, see Chapter 61, "Creating Customizable Applications" and the "Customizing Applications with MDS" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. For more information about the Customization Manager, see the "Customization Manager" section in the "Extending Runtime Editing Capabilities Using Oracle Composer" chapter, and the "Manage Customizations" section of the "Introduction to Oracle Composer" chapter of the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

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 Oracle Composer-Specific MDS Configurations" chapter of *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

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 Services. See "Using the Sandbox Manager" in the "Understanding the Customization Development Lifecycle" chapter of the *Oracle Fusion Applications Extensibility Guide*.

Setup and Maintenance ...

Select this option to launch the Oracle Fusion Functional Setup Manager application. See the *Oracle Fusion Applications Common Implementation Guide*.

14.8.1 How to Secure the Administration Menu

All teams that need the Administration link need to include the privilege and the permission in their JAZN file as defined in Example 14–34. 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 14–34 Required Privilege and Permission in JAZN File

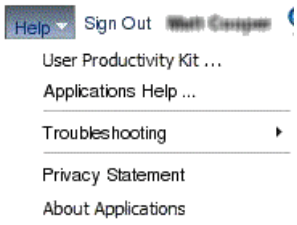
```
<app-role>
  <name>FND_ADMINISTRATION_LINK_VIEW_DUTY</name>
  <display-name>Administration Link View Duty</display-name>
  <description>Provides access to the Administration Link on the UI
Shell</description>
  <guid>EA1D0BF0BC096F11B18BDEBD5F4BDB48</guid>
  <class>oracle.security.jps.service.policystore.ApplicationRole
  </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

<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>B14A48E74ECF633A3C6E4AF95816474D</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>EA1D0BF0BC096F11B18BDEBD5F4BDB48</guid>
  </member>
 </members>
</app-role>

Permission and Grant to be included in JAZN file

<grant>
 <grantee>
  <principals>
   <principal>
     <class>oracle.security.jps.service.policystore.ApplicationRole</class>
     <name>FND_VIEW_ADMIN_LINK_PRIV</name>
     <guid>B14A48E74ECF633A3C6E4AF95816474D</guid>
   </principal>
  </principals>
 </grantee>
 <permissions>
   <permission>
     <class>oracle.security.jps.ResourcePermission</class>
     <name>resourceType=FNDResourceType,resourceName=FND_Administration_
Menu</name>
     <actions>launch</actions>
   </permission>
 </permissions>
</grant>
```

14.9 Using the Help Menu

The Help Menu, shown in Figure 14–24, provides user access to the standard help system and to troubleshooting and diagnostic tools. The menu is supplied automatically by the UI Shell and requires no developer work.

Figure 14–24 Help Menu



User Productivity Kit

The User Productivity Kit (UPK) option will be available when the UPK has been purchased, installed and configured.

This context-param entry must be added to the **web.xml** file:

```
<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" needs to be provisioned by the System Administrator. That is, an entry with `DEPLOYED_MODULE_NAME = "upk"` needs to 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 ..." would not be shown in the Help menu.

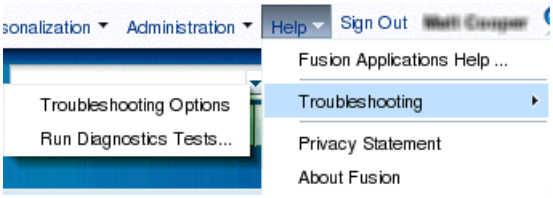
Applications Help

Select this option to launch the help system in a separate window.

Troubleshooting

When you select the Troubleshooting option, an additional menu, similar to Figure 14–25, displays.

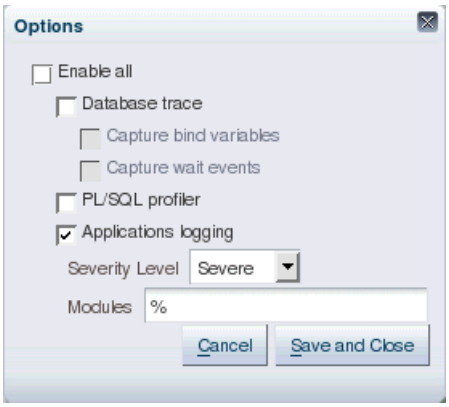
Figure 14–25 Troubleshooting Menu



- **Troubleshooting Options**

  Select this option to display the Options dialog, as shown in Figure 14–26.

Figure 14–26 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 *Oracle Fusion Applications Administrator's Guide* for information.

  Because applications logging, severity level, and modules are profiles, when users click **Save and Close**, it will insert user-specific profile values to the profiles. If users decide to revert the setting to the default site profile, they will need to 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 needs to 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 user session and initialized when a user logs into an Oracle Fusion application.

- **Database trace:**

  This option enables SQL trace for all database connections used by the current user session. See "Understanding SQL Trace and TKPROF" in the *Oracle Database Performance Tuning Guide*.

  For SQL trace, the trace file will have the FND session id appended to the end. For example, `mysid_ora_4473_881497BF7770BEEEE040E40A0D807BB1.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 also enable the SQL trace option to capture bind variables.

* **Capture wait events:**

  Select this option to also enable the SQL trace option to capture wait events.

- **PL/SQL profiler:**

  This option enables the PL/SQL hierarchical profiler for all the connections used by the current user session. See "Using the PL/SQL Hierarchical Profiler" in the *Oracle Database Advanced Application Developer's Guide*.

  For 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 command:

  ```
  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_88740EC94E3AAD2CE040E40A0D8036D8_100607104716.txt`.

  To process the collected PL/SQL profiles and view results, run `plshprof` under `$ORACLE_HOME/bin`.

- **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 "Managing Oracle Fusion Applications Log Files and Diagnostics Information" and "Troubleshooting for Oracle Fusion Applications Using Logs and Diagnostic Tests" in *Oracle Fusion Applications Administrator's Guide*.

  – **Modules:**

    Module filter for logging. This is a comma-separated list of modules to be logged. The percent sign (%) is used as a wild card. For example, **%** or **%financial%**. The percent sign (%) is the default value and, if no other value is specified, 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.

- **Run Diagnostics Tests**

Selecting this option opens the Diagnostics Dashboard user interface in a new window. For more information, see "Managing Oracle Fusion Applications Log Files and Diagnostics Information" and "Troubleshooting for Oracle Fusion

Applications Using Logs and Diagnostic Tests" in *Oracle Fusion Applications Administrator's Guide*.

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. See Section 54.3, "Setting and Accessing Profile Values."

About Applications

Select this option to display the Oracle copyright statement and information about the application.

14.10 Implementing Tagging Integration

Tagging is a service that allows users to add tags to arbitrary resources in the Oracle WebCenter so they may collectively contribute to the overall taxonomy and discovery of resources others have visited.

Tagging is a component of Oracle WebCenter. For complete information, see the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*. Specific information about tagging that you will need is in these chapters:

- "Configuring Your Application for Oracle WebCenter Web 2.0 Services"
- "Integrating the Tags Service"

This section assumes that:

- Tagging is being enabled on a business object. The Model/View already exists for the business object.
- Your pages are using the UI Shell template.
- You have identified the business objects that you want to Tag.
- You have database connections to ApplicationDB and WebCenter.
- You have enabled data security.

Important Considerations

- Tags are attached to objects, not taskflows. When you click a link belonging to a tagged object, you will navigate to a page and taskflow for viewing that particular object.
- You can enable tagging at the page level if it is clear that the page represents a specific object.
- If you have several pages in a flow, all representing the same object, you could enable tagging on every page.
- You can enable tagging in a table if each row represents a specific object.
- You can tag an object from several different places. You can give several target navigation paths from TagCenter. This allows different users to access the same object from different workareas.
- Oracle Fusion Applications Search only allows a single navigation path, but global search does allow alternate links as well as multiple Service view objects for alternate navigation paths. That is, you may tag an object from one workarea, but

on navigation from the tagged object link in TagCenter or Oracle Fusion Applications Search, the workarea that it navigates to could be different.

- Security will hide a tagged object in both TagCenter and Oracle Fusion Applications Search based on the object's data security, not taskflow security, although page and taskflow security are enforced after clicking the tag.

Preliminary Setup

The following steps are condensed from the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

1. Open your current application in JDeveloper.
2. Make sure that your database connection has access to Oracle WebCenter Schema. If it does not, create another connection to access Oracle WebCenter Schema. Name the connection `WebCenter`. *Tagging looks for this connection to access the data*.

   Note: You should now see at least two connections in your `connections.xml`: WebCenter and ApplicationDB.

3. In the resource Palette, open **My Catalog > Web Center Service Catalog > Task Flows**. Make sure that you see task flows similar to tagging-launch-dialog.
4. In the Component Palette, right-click your View Controller project and select **Project Properties > Technology Scope > Project Properties**. Ensure that Tagging Service is selected. In the Component Palette, search for *tagging*. You should see **Tagging Button** and **Tagging Menu Item**.
5. Add users and roles, and configure security and authorization for the application. Though not mandatory, it is highly recommended. Furthermore, it is required for implementing security for Tagging.

You now can enable Tagging for your business objects.

14.10.1 How to Use the Delivered Oracle WebCenter Tagging Components

Three pieces of information are needed to define Tagging to a business object:

1. The unique key of the object given which one can identify the object. This is stored in a field called RESOURCE_ID (VARCHAR2(200)) in the tagging schema. If you have multiple fields that define the unique key, use a period as the separator. For example, PK1.PK2. Additional restrictions include these:
   - There can be a maximum five columns as primary keys, such as PK1.PK2.PK3.PK4.PK5.
   - If any primary key column is null, put "null" in the concatenated Resource Id String. For example, if Resource Id is of type PK1.PK2.PK3, the value can be 123.ABC.null. *Do not* use just 123.ABC.
   - If your primary key consists of a Date, although this is highly unlikely, make sure the date value that you use in String concatenation is date formatted in Database format.
   - Teams cannot implement their own resource parser.
2. SERVICE_ID (VARCHAR2(200)): This is used to identify the object. The Applications standard is to use the logical business object name.
3. NAME

(VARCHAR2(200)): This is what will be displayed as the resource that is tagged in the Tag Center, and that will be visible to the end users when they search for a tagged item. Give it a meaningful name, such as <PO Number>+<PO Title> or Invoice Description or Customer Name.

Note: All fields are varchar2(200). Make sure that you are not violating the constraint. Also note that if, for instance, your product has three business objects that you are planning to tag, you will build three different services with the proper business object name as the service id.

14.10.1.1 Tagging a Resource (Business Object)

Follow these steps to tag a resource:

1. From the Component Palette, select **WebCenter Tagging Service**. Drag and drop the Tagging Button onto the page. (If you do not find the button, make sure you added the **WebCenter Tagging** JSP Tag Library to your user interface project.)
2. Open the Property Inspector for the Tagging Button. Enter the bound values for ResourceId, ResourceName and ServiceId, similar to Example 14–35.

Example 14–35 Entering Property Information for Tagging Button

```
<tag:taggingButton
resourceId="#{row.AuctionHeaderIdString}"
resourceName="#{row.AuctionTitle}"
serviceId="oracle.apps.pon.auctionheadersall"/>
```

Note: Make sure all the values are of type **String**.

---

3. From the Resource Palette, under **My Catalogs > WebCenter Services Catalog > Task Flows**, drag and drop the Tagging Dialog (as a Region). If you do not find the Tagging Dialog, make sure you added the **WebCenter Tagging Service View** to your user interface project (**Properties > Libraries and classpath**). Note that you may drop multiple tagging buttons on your page depending upon your requirement. You need to drop the tagging dialog only once on the page. Whenever you click a tag icon (tagging button) on the page, it will call the same tag dialog region.

Note: If placing tags within rows of a table, this region must be dropped *outside* of the table. Otherwise, it is instantiated for every row, which will not work.

The ability to tag an object is now enabled on the page. The code will look similar to that shown in Example 14–36.

Example 14–36 Enabling Tagging

```
<af:region
value="#{bindings.tagginglaunchdialog1.regionModel}"
id="taggi1"/>
```

If you have multiple objects to tag, there will be multiple tag buttons you will drop on your page, whereas there will be only one tagging dialog.

Tagging is enabled, but to see the Tagged resource in the Tag Center, you must create a service definition.

To create a service definition:

1. Expand **Application Resources > Descriptors > ADF Meta-INF**. If you already have the `service-definition.xml` file, open it. Otherwise, create the `service-definition.xml` file. Important fields are:
   - **taskFlowId**: The task flow where you want to go.
   - **resourceParamList**: The list of parameters which you want to pass to task flow. For example, your task flow takes invoiceId and invoiceType. If the RESOURCE_ID for the tagged item is 123.C45 where 123 is the invoice ID and C45 is the invoiceType, in resourceParamList you should specify invoiceId;invoiceType. The Applications Core class will parse the resource id 123.C45 and pass invoiceId=123,invoiceType=C45 to the task flow. To use a different delimiter, use the `customDelimiter` attribute.
   - **taskParametersList**: These are parameters that can be passed to the task flow in addition to the resourceParamList.
   - **navTaskKeyList**: *Do not* specify this if it is not used. Otherwise, task flows from TagCenter will always go to a specific tab while the same taskfow opened from the Tasklist or API will go to a different tab.
   - **navTaskLabel**: If using tabbed workareas, this is needed to give a title to the tab that opens showing the tagged object. An Expression Language expression can be used. It will be evaluated on page load, so it can be an Expression Language expression that the landing page can resolve.
   - **pageResourceParamList**: If the RESOURCE_ID for the tagged item is, for example, "EastCoast.C45" where EastCoast will be a page level parameter called `Region` and a taskflow parameter called `InvoiceType`. It is assumed an object will always need both composite keys to be identified as a unique entity. So, the resourceParamList will always contain the same number of parameter names as the composite keys. The taskflow must take both EastCoast and C45 as parameters. But the page level parameters can be called out in the `pageParametersList`. For example:

     ```
     resourceParamList = "Region,InvoiceType"
     pageResourceParamList = "Region"
     ```

   - **pageParametersList**: Additional page parameters where the value is static. Example: `pageParametersList = "Campaign=Sales"`
   - **customDelimiter**: This is optional. The default is a ".". If you want something different, add this parameter.

     For ease of deployment, service definition files will be stored in Oracle Metadata Services (MDS). Add the `service-definition.xml` file to the Metadata Archive (MAR) file definition.

     A sample `service-definition.xml` is provided in Example 14–37.

Example 14–37 Sample service-definition.xml

```
<service-definition id="FND_DEMO_DOC_TAG" version="11.1.1.0.0"  >
    <resource-view taskFlowId="/WEB-INF/task-flow-1.xml#task-flow-1" >
    <parameters>
        <parameter name="viewId"
                   value="RevenueWorkArea"/>
```

```
        <parameter name="webApp"
                   value="ProjectsFinancials"/>
        <parameter name="pageParametersList"
                   value="p1=viewContext"/>
        <parameter name="navTaskKeyList"
                   value="ViewRevenueItem"/>
        <parameter name="taskParametersList"
                  value="a=1;b=2"/>
        <parameter name="resourceParamList"
                    value="invoiceId;invoiceType" />
        <parameter name="navTaskLabel"
                   value="Details"/>
        <parameter name="customDelimiter"
                   value="," />
        <resource-type-key>PAGE_OBJECT_NAME</resource-type-key>
    </parameters>
    </resource-view>
    <name-key>CUSTOM_NA_KEY_TYPE</name-key>
    <description-key>CUSTOM_TY_DESCRIPTION_KEY</description-key>
</service-definition>
```

You want to put the `service-definition.xml` file in a standardized location so there are no conflicts. Create or copy the file, which by default is located at `.adf/META_INF/service-definition.xml`, to the new standardized location. There are two goals for where to put the `service-definition.xml`:

- Make it unique so two teams are not trying to push files to the exact same location in MDS.
- Standardize it to be in `meta/oracle/apps/meta`. The `/oracle/apps/meta` makes the location specific to Oracle applications. The parent `meta/` directory is used for MAR selection. The directory structure and contents then resemble:

  ```
  meta/ (this directory plus a specific sub-directory structure goes into
  the MAR)
       oracle/apps/meta/ (namespace unique to MDS)
            <product-specific-directory-structure>/ (each team has a
  unique name)
                 service-definition.xml (located in the product
  directory)
  ```

  If you use an application-level or project-level `service-definition.xml` file, you do not need to make any changes so long as your name will be unique to other applications and projects. You should never have two entries for the same SERVICE_ID, whether within the same `service-definition.xml` file or in separate `service-definition.xml` files.

  That is, the `service-definition.xml` file should be under the data model project that contains the entity objects and view objects used for the detail page. If the view object is:

  `oracle.apps.scm.receiving.receipts.receiptSummary.protectedUiModel.view.ReceiptSummaryHeaderVO`,

  the `service-definition.xml` location will be:

```
oracle.apps.scm.receiving.receipts.receiptSummary.pro-
tectedUiModel.meta.oracle.apps.meta.scm.receiv-
ing.receipts.receiptSummary.service-definition.xml.
```

2. Add the directory, such as `meta/oracle/apps/meta/<lba>/<product>/`, into the MAR.
   - Open **Application > Application Properties > Deployment**. Select the MAR file and choose **Edit**.
   - In the Edit dialog, select **User Metadata** and choose **Add**. Browse to select the meta directory you just added and click **OK**.
   - Select the meta directory (under `oracle/apps/`) and click **OK**.
   - Click **OK** in the Edit MAR dialog.
3. Add the namespace path `/oracle/apps/meta` to `adf-config.xml`, as shown in Example 14–38.

Example 14–38 Adding the Namespace Path to adf-config.xml

```
<adf-config
  <adf-mds-config xmlns="http://xmlns.oracle.com/adf/mds/config"
                  version="11.1.1.000">
    <mds-config xmlns="http://xmlns.oracle.com/mds/config" version="11.1.1.000">
       <persistence-config>
          <metadata-namespaces>
             <namespace path="/oracle/apps/meta"
              metadata-store-usage="WebCenterFileMetadataStore"/>
          </metadata-namespaces>
          <metadata-store-usage id="WebCenterFileMetadataStore"
             default-cust-store="true" deploy-target="true">
             <metadata-store
class-name="oracle.mds.dt.persistence.stores.file.SrcControlFileMetadataStore">
                  <property name="metadata-path" value="../../mds"/>
             </metadata-store>
          </metadata-store-usage>
       </persistence-config>
    </mds-config>
  </adf-mds-config>
</adf-config>
```

4. Add the entry shown in Example 14–39 to your `adf-config.xml` to enable the default resource action handler from Applications Core:

Example 14–39 Enabling the Default Resource Action Handler

```
<wpsC:adf-service-config>
    <resource-handler
class="oracle.apps.fnd.applcore.tags.handler.FndResourceActionViewHandler"/>
</wpsC:adf-service-config>
```

   Add this instruction in the header:

   ```
   xmlns:wpsC="http://xmlns.oracle.com/webcenter/framework/servi
   ce"
   ```

5. Run the page.
   - The tag icon will appear on the page. When you click the Tag icon, it will take you to the UI where you can enter a new Tag.

- You can share the tag or not share it. Only shared tags are available to Oracle Fusion Applications Search.
- After creating the tag for the first time, again hover the mouse over the icon. It will display **My Tags** and **Popular Tags**.
- Clicking the Tag takes you to the Tag Center UI where you are shown all resources that have been tagged by that word.
- Clicking a resource from TagCenter will navigate you to the task flow defined in your service definition, passing the parameters you specified so you can view the object.

14.10.1.2 Enabling Multiple Navigation Targets

Searchable and taggable objects are defined at the view object/logical business object level. The same view object/business object can be viewed and tagged in more than one workarea and taskflow. The requirement is that all users must be able to navigate from TagCenter to a detail taskflow for which they have privileges. If this taskflow is available in the current workarea, use this target before any other.

This is implemented by supplying multiple navigation targets in the current `service-definition.xml`. Each target will have its parameter separated by a caret (^).

Example 14–40 shows how to define a list of three targets:

Example 14–40 Defining a List of Three Targets

```
<resource-view

taskFlowId="/WEB-INF/dummy^/WEB-INF/dummy^/WEB-INF/Test1Frag2TF.xml#Test1Frag2TF">
  <parameters>
    <parameter name="viewId" value="DummyView^Region6UIShellPage^Test2"/>
    <parameter name="webApp"
      value="ApplCoreCRMDemo^ApplCoreCRMDemo^SimpleApplication1_application1"/>
    <parameter name="pageParametersList" value=""/>
    <parameter name="taskParametersList" value=""/>
    <parameter name="resourceParamList" value="val"/>
    <parameter name="navTaskLabel" value="My Employee Details^My Employee Details
2^My Employee Details"/>
    <parameter name="applicationStripe"
       value="ApplicationsCommon^ApplicationsCommon^AppStripe1"/>
    <parameter name="pageDefinitionName"

value="pageDef.not.exist^oracle.apps.view.pageDefs.Region6UIShellPagePageDef^oracl
e.apps.view.pageDefs.Test1PageDef"/>
   </parameters>
</resource-view>
```

The lists of `taskFlowId`, `viewId`, `webApp`, and so on, are specified as a delimited string, with each value delimited by a caret. Two parameters, `applicationStripe` and `pageDefinitionName`, are available for checking security when the target is in a different webApp.

If the current view id matches any one of the viewId values in the target list, you take the corresponding task flow (that is, if the third viewId value matches the current view id, you take the third taskFlowId value) and launch it in the current page, if the user has view access to that task flow, which overrides the order of the list. Otherwise, you check a list of link targets for the first target to which the current user has access. Application teams are responsible for making sure that all users who can access this

object have at least one match to a target taskflow defined in the `service-definition.xml`.

When the target is in a different webApp, the security check is performed by calling `checkBulkAuthorization API`. Therefore, using a standalone WebLogic Server and LDAP policy store are required. This requirement is optional when the lists of targets are within the same webApp.

14.10.1.3 Tagging a Resource at the Row Level of a Table

To add row-level tagging to a table, add a new empty column to hold the tag button/link.

All other steps remain the same as described in Section 14.10.1.1, "Tagging a Resource (Business Object).".

14.10.1.4 Searching for a Tag

The Applications Standard recommends that you do not add the Tag Search in your page. The standard way is to launch the Tag Center UI by clicking the Tag icon, or from Tags in the UI Shell global area and do the search there.

14.10.1.5 Resource Viewer for Tagged Items

It can be handy for a user to be able to click a tag and open a document. To do this, Web Center Tagging needs to know a resource viewer (a task flow) where it should take the user to show the required information.

Each development team will build a task flow where they can take the user for that resource and show the desired additional information.

1. Select or create a new task flow that will act as your resource viewer for a service (business object).
   a. In the task flow definition, define an input parameter that is called resourceId (in this example), class equal to java.lang.String, value equal to #{pageFlowScope.resourceId}, and required enabled.

      Note that the value for resourceId is set automatically when clicking a tagged item link in TagCenter.

   b. Create a method with a parameter of object id in the application module for a tagged item (named, for instance, setCurrentRow) to set the view object row based on the passed resource id parameter.

   c. Add the setCurrentRow method as the default activity in the task flow and bind the method (right click on the method in the task flow and go to page definition) with the parameter value of #{pageFlowScope.resourceId}, type java.lang.String, and name equal to the parameter variable name in the setCurrentRow method signature.

   d. Add to the new task flow the bounded task flow for the details/information page of the tagged item.

   e. Add a control flow case from the setCurrentRow method to the details/information page task flow, as shown in Figure 14–27.

Figure 14–27 Adding A Control Flow to Details Task Flow

2. Register the new task flow in the `service-definition.xml` file. You will register the resource viewer for a particular service (business object) to its section in the service definition file. See Example 14–37 for samples of the service-definition and resource-view entries.

Basically, you have defined a task flow that takes the resource id as input. Use the resource id to uniquely identify the business object and display any desired extra detail. Note that clicking a tagged item in TagCenter displays the details/information page for the tagged item in the local area of the workarea page for that task flow.

14.10.2 Implementing Tagging Security

By default, tagging does not provide any security. To avoid this problem, the development teams will implement security for each service (business object) for which tagging is enabled.

- For a business object, first implement Oracle Fusion Data Security.
- Add the `authorizerClass` and the `dataSecurityObjectName` parameter to the `service-definition.xml` file, as shown in Example 14–41.

Example 14–41 Registering a New Class in service-definition.xml

```
<service-definition id="FND_DEMO_DOC_TAG" version="11.1.1.0.0"  >
    <resource-view taskFlowId="/WEB-INF/task-flow-1.xml#task-flow-1" >
     authorizerClass="oracle.apps.fnd.applcore.tags.util.FndTagSecurity"/>
    <parameters>
       <parameter name="dataSecurityObjectName" value="FND_CRM_CASES"/>
       <parameter name="dataSecurityPrivilegeName" value="read"/>
```

FND_CRM_CASES is the object you want to secure that is found in the FND_OBJECTS table. This is normally the object's main table name.

If the `dataSecurityPrivilegeName` parameter is not set, it defaults to **read** privilege. This attribute is used in cases where a single table has different privileges for different users.

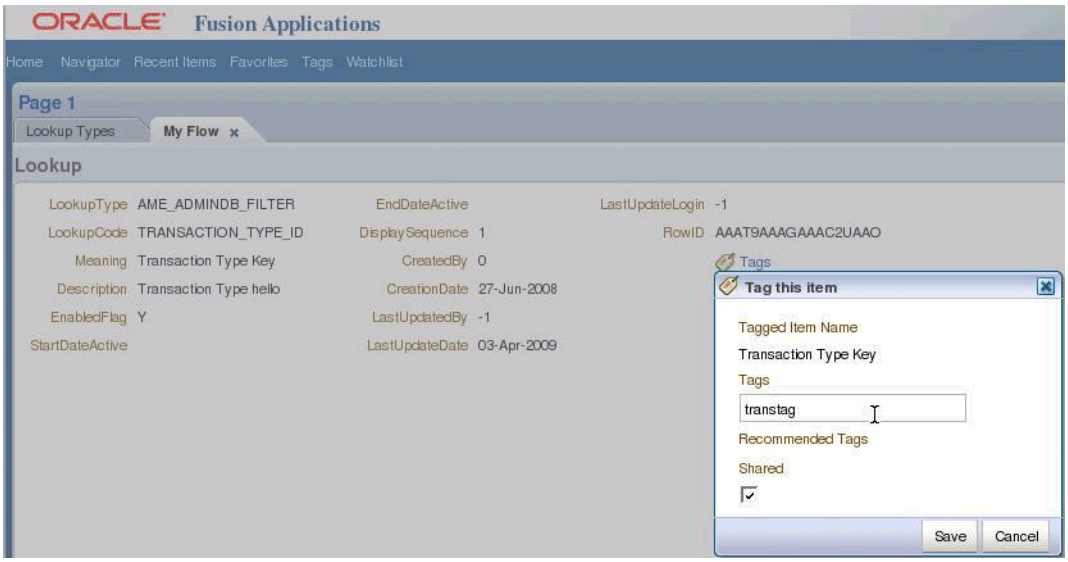
- Put the service definition into MDS. For ease of deployment, service definition files will be stored in MDS.

14.10.3 How to Use Tagging in a UI Shell Application

This section presents examples of how tagging appears in the UI Shell.

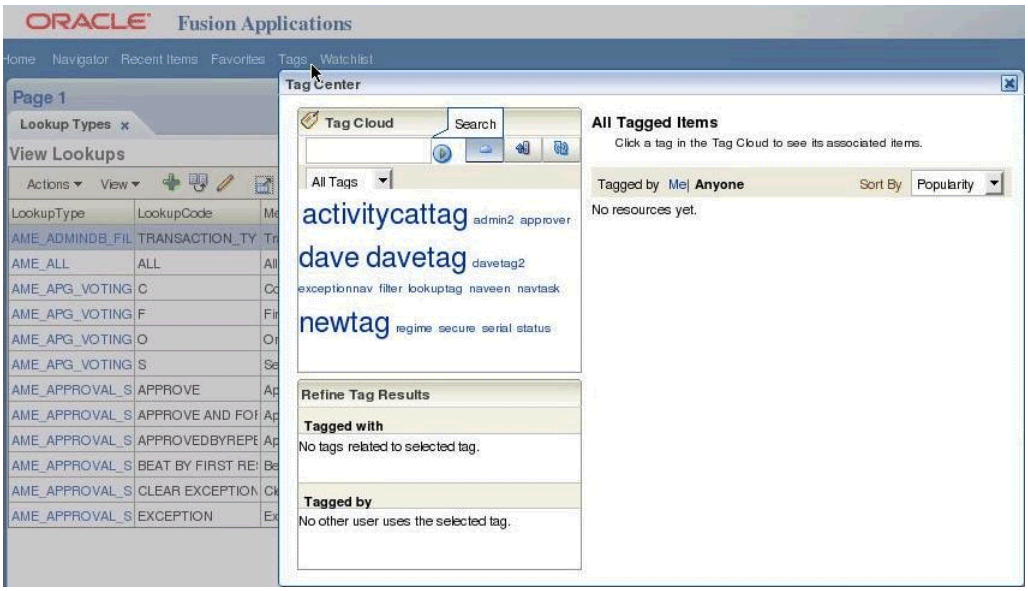
- Tagging an object by having the tagging button and tagging dialog in the task flow. On clicking the tagging button, a tagging dialog displays and prompts the user to tag the object with a name, as shown in Figure 14–28.

Figure 14–28 Tagging an Object with a Name



- To see the tagged object in the tag center flow, click the Tags link in the UI Shell global area and the tag center flow displays the list of tags available, as shown in Figure 14–29.

Figure 14–29 Displaying the List of Available Tags



- Click one of the tags in the tag cloud region of the tag center to view the tagged items, as shown in Figure 14–30.

Figure 14–30 Viewing Tagged Items

- Click the tagged item to view the object in the task flow mentioned in the service definition file, as shown in Figure 14–31.

Figure 14–31 View Object Listed In Service Definition File

- To check whether or not the object is already tagged, hover over the tagging button to see the tags. On clicking the tag link on the hover dialog, the tag center flow will be launched with the selected tag, as shown in Figure 14–32.

Figure 14–32 Launching Flow with Selected Tag

14.11 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 relaunched 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, product teams must configure the user session and ADF Security. See Chapter 47, "Implementing Application User Sessions." Without security enabled, recent items will not be captured, because the data is recorded for each authenticated user.

14.11.1 How to Choose Labels for Task Flows

Recent Items records the task flow labels for a launched task flow. Therefore, product teams 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 Section 14.2.3.4, "Supporting No-Tab Workareas").

14.11.2 How to Call Sub-Flows

`openMainTask` is used to open a new task in the Main Area of Oracle Fusion web applications that use UIShell 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`.

14.11.2.1 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 launching a task from a tasks list, product teams need to explicitly notify the UI Shell for sub-flow calls.

When `openSubTask` is called before a sub-flow is launched, 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 launched directly from the selection of menu items on Recent Items.

Note that registering sub-flows to Recent Items is optional. The decision is up to a product team's product manager.

Implementation

This API is exposed as the Data Control methods `FndUIShellController.openSubTask` and `FndUIShellController.closeSubTask` that developers will drag and drop to their page fragments to create links to notify UI Shell. The FndUIShellController Data Control is automatically available to all Oracle Fusion applications that reference Applications Core libraries.

Example 14–42 shows the signature and Javadoc of the method.

Example 14–42 Recent Items API

```
/**
* Notify UIShell to record 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's 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 launched.
*
* @param methodParameters For future implementation. No-op for now.
* @return For internal Contextual Event processing
*/
public FndMethodParameters closeSubTask(FndMethodParameters methodParameters)
```

All the parameters required to be passed in `openSubTask` are exactly same as used by the Section 14.16, "Introducing the Navigate API.".

14.11.2.2 openSubTask API Labels

The `openSubTask` API accepts the same set of parameters as used by the Section 14.16, "Introducing 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.

14.11.2.3 Launching from Recent Items

Whatever task flow details are registered while invoking the `openSubTask` API will be used by Recent Items to launch it. Recent Items takes care of launching the task flow in the right work area and web application. Product teams do not need to do anything for that. Because the `openSubTask` API supports `parametersList`, product teams can pass some requirement-specific values to it while registering their task flow to Recent Items. On launching, those passed values are available in the `pageFlowScope`. So, product teams can analyze these values and make decisions, such as if they need to first initialize the parent flow, or if they need to set `Visible` to `False` on some of the actions on the page.

14.11.3 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 launched. `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 via parent flow, even though they are launched 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 launched 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 needs to be piggybacked on the parent flow when it is launched 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 launched 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 launched 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 launched in the appropriate context.

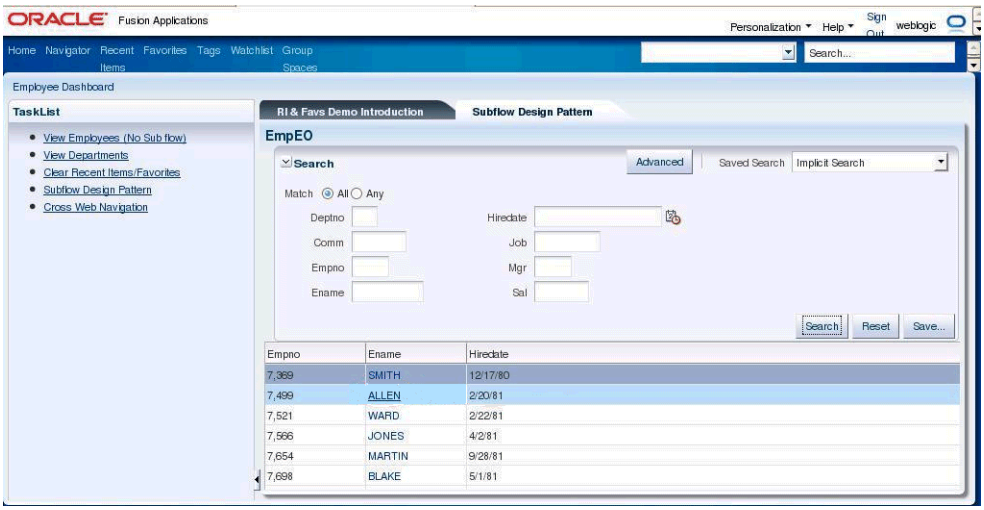
Note that the design pattern also requires the application to be able to navigate back to the parent flow from the sub-flow. The initialization code should take this into consideration, such as by setting up states to allow the sub-flow to navigate back.

In this example, you will use an Employee sample implementation to demonstrate the details of this design pattern.

Sub-flow Sample Application

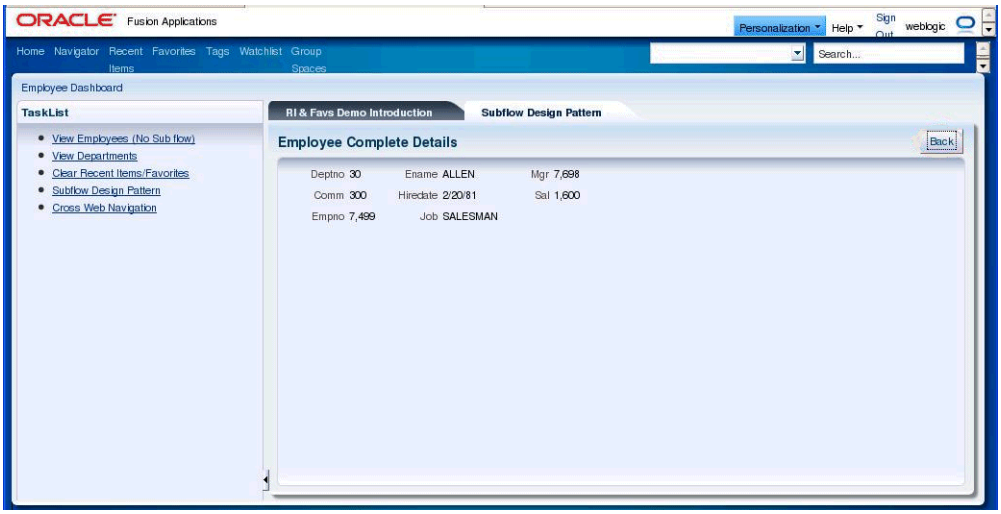
As shown in Figure 14–33, users select **Subflow Design Pattern** from the task list. They then specify some criteria to search for a specific employee or employees. From the list, they can choose the employee for whom they want to show the details.

Figure 14–33 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 14–34.

Figure 14–34 Example Employee Complete Detail Page

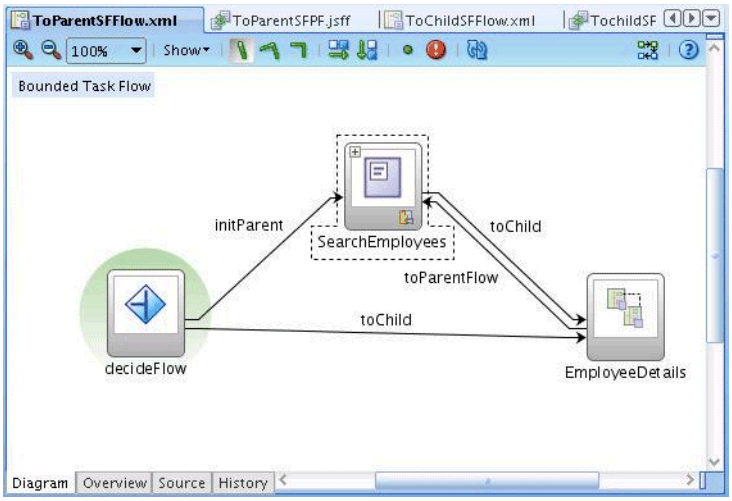


If users would like to add this Employee Complete Detail page of a specific employee to their Recent Items, product teams need to set up something extra to make this happen. If this page (actually a bounded task flow whose default page is displayed) has been bookmarked, the next time users can click it on the Recent Items list and launch it directly by skipping the search step.

14.11.3.1 Implementing the Sub-flow Design Pattern

The parent task flow named `ToParentSFFlow` is shown in Figure 14–35.

Figure 14–35 Example Parent Task Flow



`decideFlow` is the router activity that decides whether the control flow should go to the original parent flow path (`initParent`) or to the sub-flow path (`toChild`). The condition used is defined as:

```
<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 or not the `Empno` variable in the parent flow's pageFlowScope is null. `#{pageFlowScope.Empno}` is set using its input parameter `Empno` when the parent flow is called. The input parameters on the parent flow (that is, `ToParentSFFlow`) is defined as:

```
<input-parameter-definition>
  <name>Empno</name>
  <value>#{pageFlowScope.Empno}</value>
  <class>java.lang.String</class>
</input-parameter-definition>
```

When the parent flow is launched from the task List, the `Empno` parameter is not set (that is, it is not defined in the application menu's itemNode). Therefore, it is null and the router will route it to the `initParent` path.

When the sub-flow is recorded through the `openSubTask` API, `Empno` is set on the `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#ToPar
entSFContainerFlow"/>
     <NamedData NDName="parametersList" NDType="java.lang.String"
                NDValue="Empno=#{row.Empno}"/>
     <NamedData NDName="label" NDType="java.lang.String"
                NDValue="#{row.Ename} complete details"/>
     <NamedData NDName="keyList" NDType="java.lang.String"/>
     <NamedData NDName="taskParametersList" NDType="java.lang.String"/>
     <NamedData NDName="viewId" NDType="java.lang.String"
                NDValue="/DemoWorkArea"/>
     <NamedData NDName="webApp" NDType="java.lang.String"
                NDValue="DemoAppSource"/>
     <NamedData NDName="methodParameters"

NDType="oracle.apps.fnd.applcore.patterns.uishell.ui.bean.FndMethodParameters"/>
</methodAction>
```

You also set up:

- `taskFlowId` to be the parent flow's, not the sub-flow's
- `label` to be the sub-flow's

When end 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:setActionListener from="#{row.Empno}"
                          to="#{pageFlowScope.Empno}"/>
  </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` needs to be called only from the original parent flow path (that is, `initParent`), not from the sub-flow path (that is, `toChild`).

`EmployeeDetails` activity in Figure 14–35 is a Task Flow Call activity that invokes the `ToChildSFFlow` sub-flow. Before the sub-flow is executed, you need to add initialization steps. These initialization steps could include, but are not limited to:

- Set up parent states. For this example, you need to set the selected employee's row to be the current row.
- Set up the contextual area state.
- Set up states to allow the sub-flow to navigate back to the parent flow.

There are two approaches to set up the initialization steps:

- In the parent flow
- In the sub-flow

For the first approach, you can add logic to initialize both paths before the task flow call activity in the parent flow. For the second approach, you initialize states in the sub-flow by using input parameters of the sub-flow. For example, the sub-flow will take an input parameter named `Empno`. In effect, the second approach just postpones the initialization to the sub-flow.

The definition of input parameters in the Task Flow Call activity is:

```
<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 flow path, it is set to be `#{row.Empno}` using the `setActionListener` tag. For the sub-flow path, it is set using the parent flow's input parameter Empno. On the sub-flow, you need to specify its input parameters as:

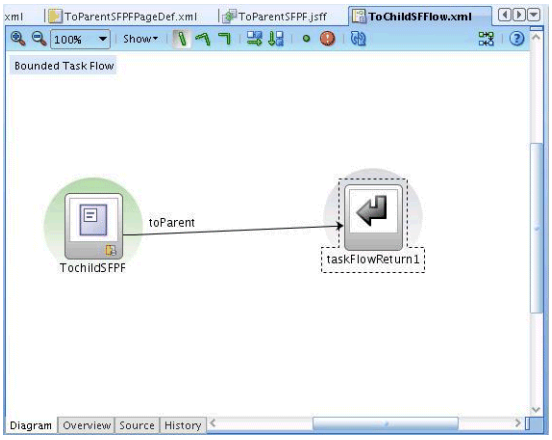
```
<task-flow-definition id="ToChildSFFlow">
   <default-activity>TochildSFPF</default-activity>
   <input-parameter-definition>
```

```
      <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`) needs to be the same as the parameter name defined on the Task Flow Call activity. When the parameter is available, ADF will place it in `#{pageFlowScope.Empno}` to be used within the sub-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-flow).

The definition of the sub-flow is shown in Figure 14–36:

Figure 14–36 Example Sub-flow Definition



In the sample implementation, you chose to implement the initialization step in the sub-flow. `Empno` is passed as a parameter to the sub-flow and used to initialize the parent state. When the sub-flow is launched, the default view activity (`ToChildSFPF`) displays. Before it renders, the `initPage` method on the ChildSFBean will be executed. The page definition of the default page is defined as:

```
<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>
```

`initPage` is specified in the executables tag and will be invoked when the page is refreshed. The `initPage` method itself is defined as:

```
public void initPage()
{
    FacesContext facesContext = FacesContext.getCurrentInstance();
    ExpressionFactory exp = facesContext.getApplication().getExpressionFactory();
    DCBindingContainer bindingContainer =
      (DCBindingContainer)exp.createValueExpression(

facesContext.getELContext(),"#{bindings}",DCBindingContainer.class).getValue(faces
Context.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);
         }
     }
}
```

`initPage` 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.

14.11.4 How to Use Additional Capabilities of Recent Items

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. This is a case that a user may have to perform often. 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 launching it from Recent Items. This is similar to using `parametersList` to specify search parameters while registering the search flow. While launching, a little programming can be done to retrieve the search parameters and execute the query with the parameter values.

Once 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 Favorites' folders. Currently, only a one-level folder is supported. Similar to Recent Items, tasks on the Favorites list can be launched directly from the menu. So, the description in this section for Recent items applies similarly 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.

14.11.5 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.

14.12 Implementing the Watchlist

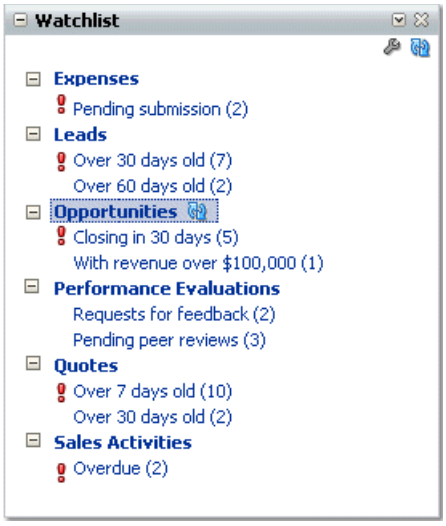
The Watchlist is a portlet/menu, accessible to Oracle Fusion Applications users, that provides a summary of items that a user wants to track. The Watchlist includes seeded items (items that are provided out of the box) categorized by functional areas, and items created by the user. Technically, the Watchlist presents a list of pre-queried searches (saved searches or standard queries) of things the user needs to track. Each item is comprised of descriptive text followed by a count. Each item also is linked to a page in a workarea 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 the users as soon as they login, 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 watchlist or individual categories as needed. Users will be able to personalize the Watchlist to hide or show items.

Figure 14–37 shows an example of the Watchlist portlet and menu.

Figure 14–37 Example Watchlist Portlet and Menu



Implementing teams 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 Watchlist category and item meaning.
- Seed information to tell the Watchlist what counts to track, how to display to 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.

14.12.1 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 developers 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 Section 14.12.4.3.1, "Summary Tables."

14.12.2 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 14–5, " ATK_WATCHLIST_CATEGORIES".
- ATK_WATCHLIST_SETUP: Represents a type of count that a Watchlist item can track. The primary key is a Watchlist item code. See Table 14–6, " ATK_WATCHLIST_SETUP".

Table 14–5 ATK_WATCHLIST_CATEGORIES

Column Name	Datatype	Required	Comments
WATCHLIST_CATEGORY_CODE	VARCHAR2(100)	Yes	PK - Unique code based on Product code Prefix. Ensure that this code begins with <PRODUCT SHORT CODE>_, so that it does not overlap with other others.
CATEGORY_LOOKUP_TYPE	VARCHAR2(30)	Yes	Reference to FND_STANDARD_LOOKUP_TYPES.LOOKUP_TYPE Product teams to seed lookup type with meaning for category (VIEW_APPLICATION_ID = 0 and SET_ID = 0). The translated lookup type meaning is shown in the Watchlist UI for category.
OWNING_MODULE_NAME	VARCHAR2(4000)	Yes	Reference to FND_APPL_TAXONOMY.MODULE_NAME for the owning product/module. This is used for seed data purposes.
OWNING_MODULE_ID	VARCHAR2(32)	Yes	Reference to FND_APPL_TAXONOMY.MODULE_ID for the owning product/module. This is used for seed data purposes.
OWNING_APPLICATION_ID	NUMBER	Yes	Reference to FND_APPL_TAXONOMY.ALTERNATIVE_ID for the owning product/module. This is used for seed data purposes.

Table 14–5 (Cont.) ATK_WATCHLIST_CATEGORIES

Column Name	Datatype	Required	Comments
REFRESH_SERVICE_ ENDPOINT_KEY	VARCHAR2(60)	Yes	This is the key to determine the host, port, context root, etc. to construct the URL for service endpoint (wsdl location). This will be based on the Applications Core lookup API that will be used for determining the end point.
REFRESH_SERVICE_NAME	VARCHAR2(400)	Yes	This is the service that needs to be invoked for count calculation (for refreshing this category).
REFRESH_SERVICE_ METHOD_NAME	VARCHAR2(400)	Yes	This column is obsoleted. The hard-coded method name will be `refreshWatchlistCategory`. Product teams will create this hard-coded service method.
ENABLED	VARCHAR2(1)	Yes	Defaults to Y. This defines if this Watchlist category is enabled/active.
CREATED_BY	VARCHAR2(64)	Yes	Standard WHO Column
CREATION_DATE	TIMESTAMP	Yes	Standard WHO Column
LAST_UPDATED_BY	VARCHAR2(64)	Yes	Standard WHO Column
LAST_UPDATE_DATE	TIMESTAMP	Yes	Standard WHO Column
LAST_UPDATE_LOGIN	VARCHAR2(32)	Yes	Standard WHO Column

Table 14–6 ATK_WATCHLIST_SETUP

Column Name	Datatype	Required	Comments
WATCHLIST_ITEM_CODE	VARCHAR2(100)	Yes	PK - Needs to use Category Code Prefix
ITEM_LOOKUP_CODE	VARCHAR2(30)	Yes if WATCHLIST_ ITEM_TYPE != USER_ SAVED_SEARCH	Reference to FND_ LOOKUPS.LOOKUP_CODE Product teams to seed lookup code with meaning for the parent category lookup type. The translated lookup meaning is shown in the Watchlist UI with the count appended.
CATEGORY_CODE	VARCHAR2(30)	Yes	
PRIVILEGE_BASED	VARCHAR2(1)	Yes	Specifies if this Watchlist item is created against a security action instead of a specific user.
OWNING_PRIVILEGE_NAME		Yes if PRIVILEGE_BASED = Y	Defines if this item is created against a security action - users that have this action will be able to view this item. Required if PRIVILEGE_BASED = Y
FUNCTION_PRIVILEGE_ NAME	VARCHAR2(400)		Defines the region action for this item's drilldown workarea. This is the page definition for the drilldown view/jspx that is part of your jazn-data.xml. The user needs to have this permission policy to view the item in the Watchlist UI.

Table 14–6 (Cont.) ATK_WATCHLIST_SETUP

Column Name	Datatype	Required	Comments
WATCHLIST_ITEM_TYPE	VARCHAR2(30)	Yes	Defines the Watchlist item type - maps to 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)
HUMAN_TASK_DEF_ID	VARCHAR2(200)	Yes if WATCHLIST_ITEM_TYPE = HUMAN_TASK	Human Task Definition Identifier. Required if WATCHLIST_ITEM_TYPE = HUMAN_TASK
HUMAN_TASK_STATE	VARCHAR2(100)	Yes if WATCHLIST_ITEM_TYPE = HUMAN_TASK	Human Task State Identifier. Required if WATCHLIST_ITEM_TYPE = HUMAN_TASK
REFRESH_AGE	NUMBER(9)	Yes if WATCHLIST_ITEM_TYPE != HUMAN_TASK	Defines the age for count in seconds. After this many seconds have passed since the last refresh time, the Watchlist UI would issue a count recalculation request Required if WATCHLIST_ITEM_TYPE != HUMAN_TASK
VIEW_OBJECT	VARCHAR2(400)	Yes if WATCHLIST_ITEM_TYPE != HUMAN_TASK	Complete path of the view object that needs to be executed for count calculation. Required if WATCHLIST_ITEM_TYPE != HUMAN_TASK
SUMMARY_VIEW_ATTRIBUTE	VARCHAR2(400)	Yes if WATCHLIST_ITEM_TYPE != HUMAN_TASK and this is a summary view object	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. Required if WATCHLIST_ITEM_TYPE != HUMAN_TASK and this is a summary view object.
APPLICATION_MODULE	VARCHAR2(400)	Yes if WATCHLIST_ITEM_TYPE != HUMAN_TASK	Complete path of the application module that contains the view object instance that needs to be executed for count calculation. Required if WATCHLIST_ITEM_TYPE != HUMAN_TASK

Table 14–6 (Cont.) ATK_WATCHLIST_SETUP

Column Name	Datatype	Required	Comments
AM_CONFIG_NAME	VARCHAR2(400)	Yes if WATCHLIST_ITEM_TYPE != HUMAN_TASK	The application module Configuration Name for creating an instance of the application module from code. This is typically AMLOCAL. Required if WATCHLIST_ITEM_TYPE != HUMAN_TASK
VIEW_OBJECT_INSTANCE	VARCHAR2(400)	Yes if WATCHLIST_ITEM_TYPE != HUMAN_TASK	The Instance Name for the view object in the application module. Required if WATCHLIST_ITEM_TYPE != HUMAN_TASK
VIEW_CRITERIA_ID	VARCHAR2(400)	Yes if WATCHLIST_ITEM_TYPE = SEEDED_SAVED_SEARCH	The View Criteria that needs to be applied when executing the view object. Required if WATCHLIST_ITEM_TYPE != HUMAN_TASK
NAVIGATION_URL_KEY	VARCHAR2(60)	Yes	This is the key to determine the host, port, and context root to construct the URL for the UI drilldown for Watchlist item. This will be based on the Applications Core lookup API that will be used for UI navigation across J2EE applications.
VIEW_ID	VARCHAR2(400)	Yes	The view id (as per the UI Shell menu) for the workarea/page that contains the task flow for this Watchlist item's drilldown from the Watchlist UI.
PAGE_PARAM_LIST_STRING	VARCHAR2(400)	No	Parameters list for the page. If the target workarea page accepts page parameters, this is a semicolon-delimited String of name value pairs.
TASKFLOW_ID	VARCHAR2(400)	Yes	The task flow for this Watchlist item's drilldown from the Watchlist UI.
TF_KEY_LIST_STRING	VARCHAR2(400)	No	Key list to pass into the task flow to open in the target workspace. This is a semicolon-delimited keys or key-value pairs. For example, "key1;key2=value2"

Table 14–6 (Cont.) ATK_WATCHLIST_SETUP

Column Name	Datatype	Required	Comments
TF_PARAMETER_STRING	VARCHAR2(400)	Yes if WATCHLIST_ITEM_TYPE IN (SEEDED_SAVED_SEARCH, USER_SAVED_SEARCH)	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" For user created saved search, the view criteria id will be appended to this string (the string will need to end with <paramName>= for saved search).
TASK_TAB_LABEL	VARCHAR2(400)	Yes	Label for the task flow to open in the target workspace.
ENABLED	VARCHAR2(1)	Yes	Defaults to Y. This defines if this Watchlist category is enabled/active.
CREATED_BY	VARCHAR2(64)	Yes	Standard WHO Column
CREATION_DATE	TIMESTAMP	Yes	Standard WHO Column
LAST_UPDATED_BY	VARCHAR2(64)	Yes	Standard WHO Column
LAST_UPDATE_DATE	TIMESTAMP	Yes	Standard WHO Column
LAST_UPDATE_LOGIN	VARCHAR2(32)	Yes	Standard WHO Column

14.12.3 Supported Watchlist Items

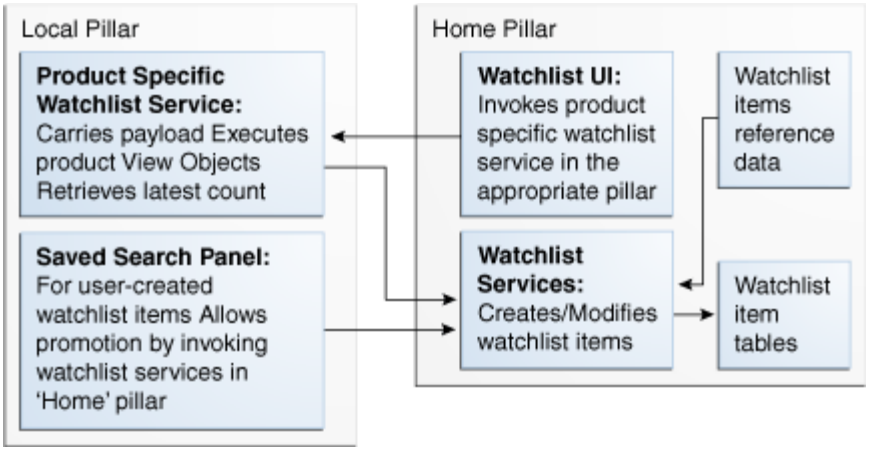
Supported Watchlist items are all *asynchronous* (that is, queries are executed on demand when the user requests a refresh or just before the Watchlist UI is shown). There are four types of asynchronous Watchlist items (`watchlist_item_type`):

- Seeded queries (SEEDED_QUERY)
- Seeded saved searches (SEEDED_SAVED_SEARCH)
- User-created saved searches (USER_SAVED_SEARCH)
- Human task-flow items (HUMAN_TASK)

14.12.3.1 Asynchronous Items Overview: Expense Reports Saved Search

For asynchronous Watchlist items, the count is only updated 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 only be updated upon request, and events that change the count will not simultaneously be updating the Watchlist. In this case, *Watchlist code is responsible for querying the count of an asynchronous Watchlist item on demand*. Figure 14–38 shows the flow of an asynchronous Watchlist item.

Figure 14–38 Asynchronous Watchlist Item Flow



For the Expense report saved search panel example:

- Since 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).

14.12.3.2 Summary of Implementation Tasks

At a high-level, for asynchronous Watchlist items, the developer tasks are:

- (Not needed for Human Task) Determine/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.

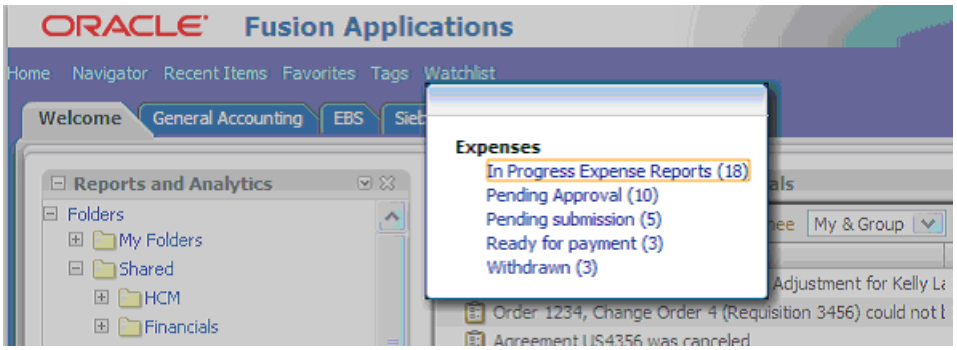
  For example, most view objects seeded for the Watchlist would need to filter by user to show counts specific to the logged-in user. You could create a view criteria with a bind variable called userId and specify the default value as a groovy expression to determine the current user id from the security 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 14–43 shows a code snippet from the view object xml for bind variable with default value.

Example 14–43 Example Code from the View Object XML for Bind Variable with Default Value

```
<Variable
    Name="userId"
    Kind="where"
    Type="java.lang.String">
    <TransientExpression><![CDATA[return
adf.context.securityContext.userName;]]></TransientExpression>
  </Variable>
```

- (Optional) Set up a Summary view object to facilitate 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 will only 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 launch the refresh process.
- Determine/code task flows for drill-down 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 query panel and work with Watchlist API. For promotion and unpromotion of user-saved searches, code will have to invoke a method in the provided Watchlist JAR, which will then handle interaction back to the Watchlist. Add promotion and unpromotion 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 14–39.

Figure 14–39 Example Watchlist Menu Drilldown



- Create FND_LOOKUPS for displayed Watchlist category and item meaning (FND_STANDARD_LOOKUP_TYPES.LOOKUP_TYPE). Product teams will need to seed the lookup type with meaning for category (VIEW_APPLICATION_ID = 0 and SET_ID = 0). The translated lookup type meaning is shown in the Watchlist UI for category, while the corresponding lookup value meanings are shown in the Watchlist UI for items (user saved search item meanings come from saved search directly). Product teams will need to create seed data for this lookup.
- Example 14–44 presents sample code to create or update the lookup.

Example 14–44 Sample Code to Create/Update Watchlist Lookup

```
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',
  X_MEANING => 'Expenses',
  X_DESCRIPTION => 'Expenses Watchlist Category',
  X_REFERENCE_GROUP_NAME => null
);
end;

declare
begin
FND_LOOKUP_VALUES_PKG.CREATE_OR_UPDATE_ROW (
  X_LOOKUP_TYPE           => 'FIN_EXM_WATCHLIST_CATEGORY',
  X_VIEW_APPSNAME         => 'FND',
  X_LOOKUP_CODE           => 'SAVED_EXPENSE_REPORTS',
  X_MEANING               => 'In Progress Expense Reports'
--  X_SET_CODE              IN VARCHAR2 DEFAULT NULL,
--  X_DESCRIPTION           IN VARCHAR2 DEFAULT NULL,
--  X_ENABLED_FLAG          IN VARCHAR2 DEFAULT 'Y',
--  X_START_DATE_ACTIVE     IN VARCHAR2 DEFAULT NULL,
--  X_END_DATE_ACTIVE       IN VARCHAR2 DEFAULT NULL,
--  X_DISPLAY_SEQUENCE      IN NUMBER DEFAULT NULL
  );
end;
```

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.

14.12.4 How to Use the Watchlist

Developers should follow these procedure to use the Watchlist.

14.12.4.1 Making the Watchlist Link in UI Shell Global Area Work

To ensure the Watchlist link works in your pages, complete these steps.

- Add this ADF library JAR file to the SuperWeb user interface project for your application (the JAR file must be part of your WAR in WEB-INF/lib):

  `fusionapps/jlib/AdfAtkWatchListPublicUi.jar`

- Add this dependent model ADF library JAR file in your application (the JAR file must be part of your EAR in APP-INF/lib):

  `fusionapps/jlib/AdfAtkWatchListProtectedModel.jar`

- Add this dependent resource bundle ADF library JAR file in your application (the JAR file must be part of your EAR in APP-INF/lib):

  `fusionapps/jlib/AdfAtkWatchListPublicResource.jar`

- Add these resource-ref entries to `web.xml` in your SuperWeb user interface project:

```
<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>
```

14.12.4.2 Seed Reference Data (All items)

Refer to Section 14.12.2, "Watchlist Physical Data Model Entities" for details of the entire Watchlist data model. Seed only ATK_WATCHLIST_CATEGORIES and ATK_WATCHLIST_SETUP.

14.12.4.3 Create Summary View Object (SEEDED_QUERY)

- By default, Watchlist code will access the application module/view object that is specified in the setup table, and rerun the query to refresh the Watchlist count. The summary view object is a way for a product team to get the count in its own way. 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 just take the first row and get the specified attribute.
- Thus, the developer task is to create a view object with the correct count in the attribute specified in the setup table.

14.12.4.3.1 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 per 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, since you are only interested 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 14–7.

Table 14–7 Example Summary Table

BUID	Count
1	3
2	3
3	3

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.

Developers can use summary tables to populate their Summary view object attribute.

14.12.4.4 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. Currently, saved searches in MDS are stored in the file system as an XML file for each view object. The developer steps to create this file are:

- Make sure you have enabled MDS for your application.
- Run the application and visit the desired search page.
- Use the UI to create saved searches and name them. For each saved search, select **Run Automatically**.
- Examine the **adf-config.xml** file to determine where your file-based MDS repository is located.
- In a terminal, change to the directory:

  ```
  <MDS
  repository>/persdef/oracle/apps/.../mdssys/cust/user/<user>/
  ```

  <user> would be the user you used to save the search with, or 'anonymous' by default.
- Inside that directory, there should be an **xxxVO.xml.xml** file that contains the created saved searches. Keep that file.
- 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.

14.12.4.5 Creating Application Module/View Objects (All except HUMAN_TASK)

Refer to Section 14.12, "Implementing the Watchlist."

14.12.4.6 Setting Up Service (All except HUMAN_TASK)

For the same Watchlist items, the Watchlist portlet needs to be able to invoke a refresh. Each product team 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, which will, in turn, 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.

14.12.4.7 Importing All Watchlist-Related AMs

Your Service project will need to import the other JAR files from your product that will need to be used by the Watchlist code.

14.12.4.8 Nesting Watchlist Application Module

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.

14.12.4.9 Using the refreshWatchlistCategory Method

```
public void refreshWatchlistCategory(String categoryCode)
```

This method, shown in Example 14–45, refreshes all Watchlist items in the corresponding category.

Example 14–45 Refreshing Watchlist Items

```
public void refreshWatchlistCategory(String categoryCode) {
    AtkWatchlistPublicAMImpl wlAM = this.getAtkWatchlistPublicAMImpl();
    wlAM.refreshWatchlistCategory(categoryCode);
}
```

14.12.4.10 Importing Watchlist JAR Files into the Saved Search Project (USER_SAVED_SEARCH)

For subsequent steps that require running Watchlist APIs from your code, you will need to import the Watchlist JAR files. These also contain an AppMasterDB connection that will need to 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.

14.12.4.11 Promoting Saved Search to the ATK Watchlist (USER_SAVED_SEARCH)

Every Watchlist-enabled saved search panel will need to 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 of his or her own 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".
- Make sure 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. Make sure 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 14–46:

Example 14–46 Creating the Backing Bean

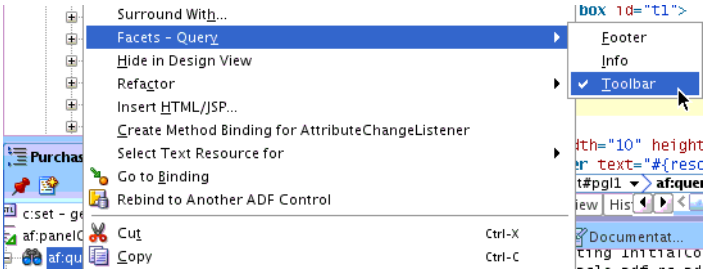
```
import oracle.adf.view.rich.model.QueryDescriptor;
import oracle.adf.view.rich.model.QueryModel;
..................

    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:
getWatchListUserSavedSearchList method: 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:
getWatchListUserSavedSearchList method: 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:
getWatchListUserSavedSearchList method: 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;
    }
```

14.12.4.11.1 How to Promote a User-Saved Search to the Watchlist Follow these steps to integrate the ATK team-provided task flow to promote saved searches to the Watchlist.

Note: If you already have implemented promoting the user-saved search to the Watchlist, see **Additional Steps for Existing Consumers**.

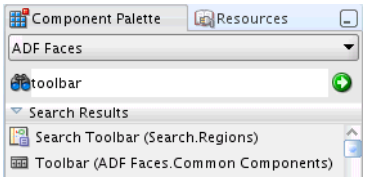
1. Make sure the **AdfAtkWathcListPublicUi.jar** file is available (usually in `fusionapps/jlib`).
2. Launch JDeveloper and open the .jspx or .jsff page containing the query region whose saved searches have to be promoted.
3. In the query region, add a toolbar facet.
   a. Right-click the component.
   b. In the menu that opens, select **Facets-Query**.
   c. From the submenu, select **Toolbar**, as shown inFigure 14–40.

Figure 14–40 Adding Toolbar Facet to Query Region



4. 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 14–41, onto the page.

Figure 14–41 Adding an ADF Toolbar Component

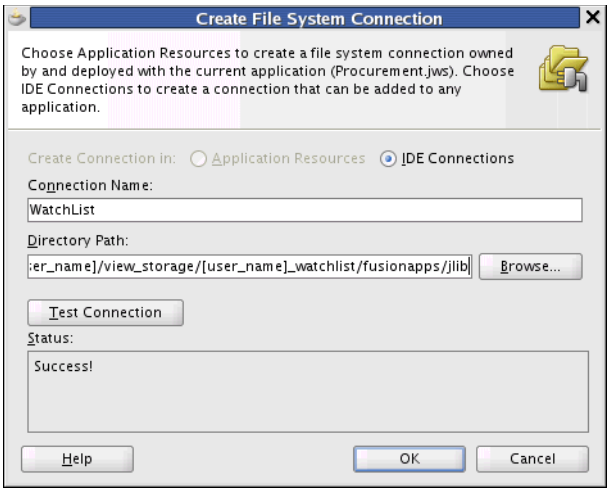


The toolbar facet in the Source view will look similar to:

```
<f:facet name="toolbar">
  <af:group id="g1">
    <af:toolbox id="t1">
      <af:region
value="#{bindings.AtkWatchlistUserSavedSearchPromotionTF1.regionModel}"
                id="r7"/>
    </af:toolbox>
    <af:toolbar id="t2"/>
  </af:group>
</f:facet>
```

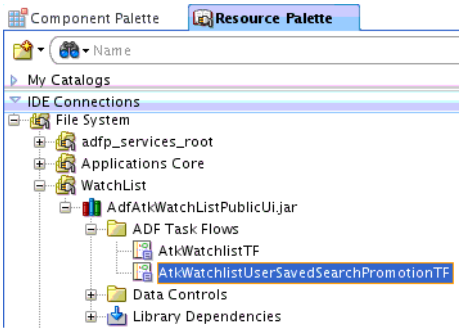
5. Open the Resource Palette and create a File System connection to the directory containing the **AdfAtkWatchListPublicUI.jar** file, as shown in Figure 14–42.

Figure 14–42 Creating the File System Connection



6. Expand the connection node and the ADF library node in the Resource Palette as shown in Figure 14–43.

Figure 14–43 Expanding the Connection Node



Once the ADF Task Flows node is expanded, you should see two task flows. The task flow **AtkWatchlistUserSavedSearchPromotionTF** is the one to be used by product teams.

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 displays. Enter the following values for the mandatory parameters.

   - **categoryCode**: Provide the WATCHLIST_CATEGORY_CODE that has been seeded in the ATK tables.
   - **watchlistItemCode**: Provide the WATCHLIST_ITEM_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 14–46. In this case, the value is `watchListUserSavedSearchList`, found in **ADF**

Managed Beans > backingBeanScope > searchOrderScheduleBackingBean > watchListUserSavedSearchList.

The Expression will be:

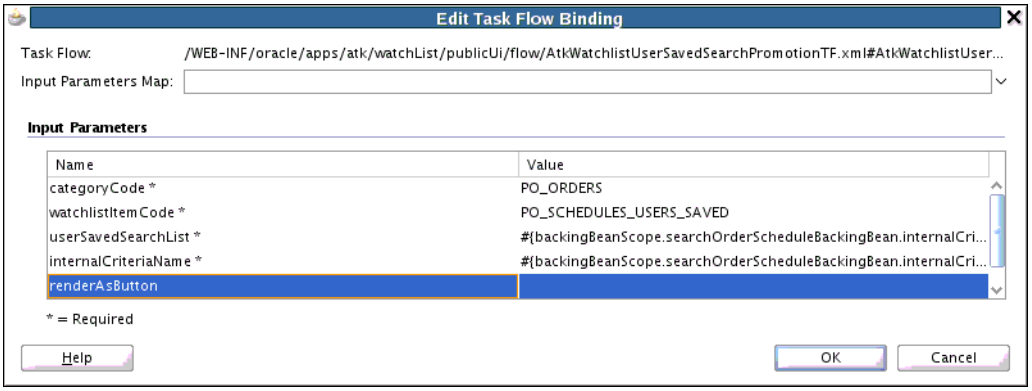
```
#{backingBeanScope.searchOrderScheduleBackingBean.watchListUserSavedSearchList}
```

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 14–44.

Figure 14–44 Completed Edit Task Flow Binding Dialog



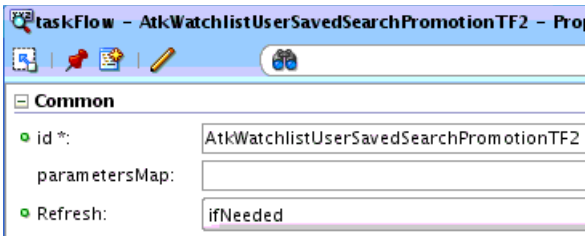
8. Click **OK**. This creates a region component in the UI page, as shown in Figure 14–45.

Figure 14–45 Created Region Component

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 14–46. 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.

Figure 14–46 Setting Refresh to ifNeeded

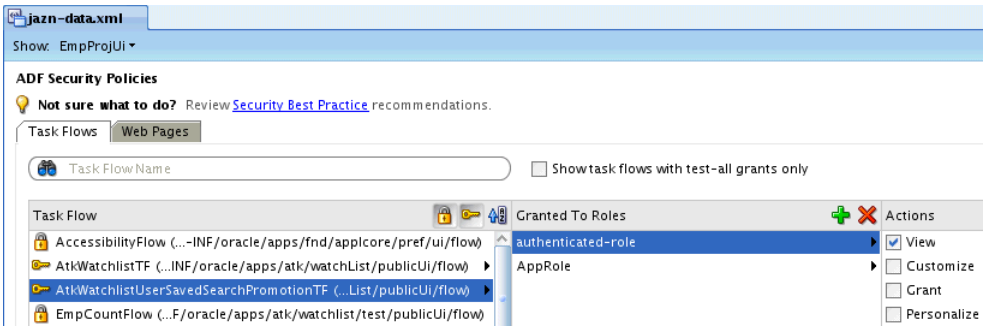


11. Add a task flow security permission to the **AtkWatchlistUserSavedSearchPromotionTF** task flow in the **jazn-data.xml** file.

    a. Open the **jazn-data.xml** file and select the **ADF Policies** tab.

    b. From the Task Flow list, select the **AtkWatchlistUserSavedSearchPromotionTF**, grant it to an appropriate role, and select appropriate actions as shown in Figure 14–47.

Figure 14–47 Adding Security Permissions to jazn_data.xml

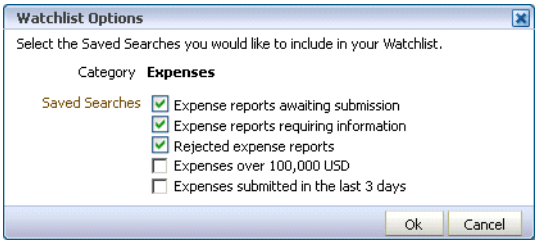


12. Run the UI page. In the toolbar facet of the query region, there will be a **Watchlist Options** button, as shown in Figure 14–48.

Figure 14–48 Watchlist Options Button in Toolbar Facet

When you click the button, a popup with the list of all saved searches displays, as shown in Figure 14–49.

Figure 14–49 List of Saved Searches



Additional Steps for Existing Consumers

For product teams that already have implemented promoting Saved Search to the ATK Watchlist, there are four additional steps.

- 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}"/>
  ```

- Use the `public List<String> getWatchListUserSavedSearchList()` Java method by passing the QueryModel binding of your `af:query`.
- 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}"
  ```

- In **someBackingBeanScopeBean.java**, such as the one you created in Example 14–46, add the two methods shown in Example 14–47.

Example 14–47 Additions to the backingBeanScopeBean

```
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);
     }
     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;
  }
```

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}`.

14.12.4.12 Code Task Flows to Accept Parameters (All except HUMAN_TASK)

If you have seeded the information properly, your normal task flows should work when drilled-down to from the Watchlist portlet.

14.12.4.12.1 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.

The developer will be concerned with implementing the two steps for the destination task flow. First, he or she can retrieve the ViewCriteriaName from the PageFlowScope with the code shown in Example 14–48.

Example 14–48 Retrieving the ViewCriteriaName from the PageFlowScope

```
Map pfs = RequestContext.getCurrentInstance().getPageFlowScope(); String vcName =
(String) pfs.get("vcName");
```

Second, the developer can use the code shown in Example 14–49 to apply the ViewCriteria to the search panel. If no ViewCriteria was passed, the code loads the default ViewCriteria.

Example 14–49 Applying ViewCriteria to the Search Panel

```
if (vcName == null || vcName.equals("")) {
// If no ViewCriteria is given, load default VC
DCBindingContainer dcbc =
(DCBindingContainer)BindingContext.getCurrent()
.getCurrentBindingsEntry();
FacesCtrlSearchBinding fcsb =
(FacesCtrlSearchBinding)dcbc
.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);
}
BindingContainer bindings =
BindingContext.getCurrent().getCurrentBindingsEntry();
OperationBinding method = (OperationBinding)
bindings.getControlBinding("applyViewCriteriaByName");
method.getParamsMap().put("name", vcName);
method.execute();
}
```

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 static variable to ensure that it only runs once.

14.12.4.13 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 that you will ultimately run will need to include these Watchlist UI JAR files: **AdfAtkWatchListPublicUI** and its dependent model JAR file **AdfAtkWatchListProtectedModel**.

14.12.4.14 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 ejb-jar.xml 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:

```
<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 web.xml in your SuperWeb project next to the similar entry for ApplicationDB. You need resource-ref entries for both ApplicationDB and AppMasterDB:

```
<resource-ref>
  <res-ref-name>jdbc/AppMasterDBDS</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

- **connections.xml** should have a valid database entry for AppMasterDB.

14.13 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.

14.13.1 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. GroupSpaces functionality attempts to retrieve the group spaces for the logged-on user. Without a secure application, this functionality would fail.

14.13.2 How to Implement Group Spaces

Follow these steps to implement GroupSpaces.

1. Make sure the Oracle WebCenter 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 Oracle WebCenter Spaces WebService. To do this:
   a. Right-click **Connections** in the Application Resource palette.
   b. Choose **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.

14.13.3 Overview of Group Spaces Functionality

The Group Spaces functionality implements these features:

- When the GroupSpaces 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 chrome but will still render all the tabs within that Group Space. This chrome level suppresses the WebCenter chrome but will still render all the tabs within that Group Space.
- When the user clicks View All GroupSpaces, the same UI as the My Group Spaces in the spaces application is rendered. This is also rendered as an iFrame within the WebCenter HomePage tab where it suppresses the chrome as well as the top level WebCenter tabs.

14.13.4 How to Pass a Chromeless Template

When navigating to the WebCenter home page, a WebCenter 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, TagCenter, 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.

14.14 Implementing Activity Streams and Business Events

Activity Streams is a feature provided by the WebCenter. Product teams 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.

14.14.1 Introduction to WebCenter Activities

A WebCenter Activity is comprised of the following:

- **Actors** - 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 a Activity but for Business Events, only the event source is used as an object.

WebCenter 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, Object Types and resource-view definitions. An ActivityType needs to 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 ResourceBundle. 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}
```

14.14.2 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 ActivitiyStream 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 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 hyper link 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 hyper link 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.

14.14.3 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 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 14–50 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 J2EE 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.

Example 14–50 BusinessActivityPublisher.java

```
/**
 * This class is responsible for publishing business events as WebCenter
Activities to
 * the ActivityStreaming service. This is a singleton per J2EE 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 J2EE 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 anew one.
     * @return
     */
    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 J2EE container, this method is called by Applications Core
     * ServletContextListener.
     */
    public void release()
```

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. Product teams 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 14–51.

Example 14–51 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();
```

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. Product teams 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 14–52.

Example 14–52 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();
```

14.14.4 How to Implement Activity Streams

This section provides details about the steps involved in integrating this feature with Oracle Fusion Applications.

14.14.4.1 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>
```

14.14.4.2 Overriding isActivityPublishingEnabled() to Enable Activity Publishing

By default, Business Events are not published as Activities. Product teams should override the `isActivityPublishingEnabled()` method to enable Activity publishing for an Entity Object. Table 14–8 shows the details about the APIs exposed in `OAEntityImpl` that product teams can override.

Note that, except for the `isActivityPublishingEnabled()` method, other methods mentioned in Table 14–8 should be avoided in favor of transient attributes specified in Section 14.14.4.3, "Defining Activity Attributes Declaratively."

Table 14–8 Overriding isActivityPublishingEnabled()

Method	Return Type	Description	Optional/ Mandatory	Corresponding Declarative Transient Attribute (See Section 14.14.4.3)
isActivityPublishingEnabled()	boolean	By default the base class implementation returns false. This will enable Activity publishing for this Entity Object.	Mandatory	
getActivityActorsGUIDs()	String []	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 when the Business Event is raised.	Optional	WCActivityActorGuid1, WCActivityActorGuid2
getActivityStreamServiceId()	String	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 is defaulted to the full name of the Entity Object.	Optional	WCActivityServiceId
getAdditionalServiceIds()	String []	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.	Optional	WCAdditionalActivityServiceId1, WCAdditionalActivityServiceId2
getActivityObjectTypeNames()	String	This API can be overridden to return multiple object type names. Value provided should be a "," 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 primary object. When creating the primary object for a Activity, object-type of the first object listed in service-definition xml should be used even though the custom attrs used to construct this object are fetched from a diff object-type based on getActivityObjectTypeNames() or WCActivityObjectTypeNames. This is necessary for follow model to work. The order of the objects in this array can be used to reference the objects in Activity message format string.	Optional	WCActivityObjectTypeNames

14.14.4.3 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 Enterprise 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 14–9.

Table 14–9 Transient Attributes that Can Be Passed with the Payload

Attribute	Type	Description	Sample Value
WCActivityServiceId	String	This attribute's value is used to identify the Business Object service to be associated with the Activity.	"oracle.apps.crmdemo.model.OpportunityEO"
WCAdditionalActivityServiceId1	String	This attribute's value is used to identify any additional service that needs to be associated with the Activity.	Values are similar to those of WCActivityServiceId
WCAdditionalActivityServiceId2	String	This attribute's value is used to identify any additional service that needs to be associated with the Activity.	Values are similar to those of WCActivityServiceId
WCActivityActorGuid1	String	This attribute's value is used to identify any actor that needs to be associated with the Activity if the default actor value needs to be overridden. For instance, in the message it can be accessed as actor[0]	"<User_GUID>"
WCActivityActorGuid2	String	This attribute's value is used to identify any actor that needs to be associated with the Activity if the default actor value needs to be overridden. For instance, in the message it can be accessed as actor[1]	"<User_GUID>"
WCActivityObjectTypeNames	String	This attribute should provide a "," 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 primary object. When creating the primary object for a Activity, object-type of the first object listed in service-definition xml should be used even though the custom attrs used to construct this object are fetched from a diff 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.	Emp, Dept

14.14.5 How to Define Activities

Defining Activities requires:

- Adding the ActivityStream UI Task Flow
- Defining Activities in `service-definition.xml`

14.14.5.1 Adding the ActivityStream UI Task Flow

To add the ActivityStream task flow:

1. Make sure your user interface project includes the WebCenter 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 "Integrating the People Connections Service" chapter in the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

14.14.5.2 Defining Activities in service-definition.xml

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:

```
<app>/<lba>/<product>/<*project*>/*meta/oracle/apps/meta*/<lba>/
<product>/service-definition.xml
```

The name must be unique, such as:

```
helpPortal/atk/helpPortal/model/meta/oracle/apps/meta/atk/helpPo
rtal/service-definition.xml
```

To define Activities in `service-definition.xml`, 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 displays, select the MAR file and click **Edit**.
2. In the Edit dialog, select **User Metadata** and click **Add**. Browse to the meta directory 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.

```
<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. Make sure the event name of the Entity Object matches the activity-type name attribute value.

```
<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 ResourceBundle or an XLIFF bundle. Oracle Fusion Applications use 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 ResourceBundles. The Common String Repository is used for the message format strings.

These attributes are supported on the activity-type element:

- `messageFormatKey` - Used on Activity Stream full view task flow.
- `summaryByListMessageFormatKey` - Used in summary view Activity Stream task flow.
- `summaryByCountMessageFormatKey` - Used in summary view task flow.

messageFormatKey

The value of this attribute points to the key defined in ResourceBundle. 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.

Sample using Java ResourceBundle:

```
<resource-bundle-class>oracle.apps.crm.OpportunityResourceBundle</resource-bund
le-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/aggregate the Actors either by listing them if there are 3 or fewer, or by counting them if there more than 3.

c. For remaining activities, summarize by finding a common Actor. Summarize/aggregate the objects either by listing them if there are 3 or fewer, or by counting them if there are more than 3.

For example, for the following activities:

a. James updated Project Alpha tasks.

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 14–53 shows sample format strings for the above scenario.

Example 14–53 Sample Format Strings for a Summarized View

```
<resource-bundle-class>oracle.apps.crm.OpportunityResourceBundle</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"
               summaryByListMessageFormatKey="OPPTY_STATUS_UPDATED_SUMMARY_LIST"
               summaryByCountMessageFormatKey="OPPTY_STATUS_UPDATED_SUMMARY_CNT"
               defaultPermissionLevel="SHARED"/>
    </generic-activity-category>
 </activity-types>
```

In 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 WebCenter 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.

```
<object-types>
   <object-type name="OpptyId"
               displayName="Opportunity Object"
               description="Opportunity Object"
```

```
                    iconURL="">
    </object-type>
</object-types>
```

7. 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 14–10 will be used.

Table 14–10 ObjectType Custom Attributes

Name	Description
service-ref-id	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 used to reference serviceId of some other Business Object.
object-id-attr	The name of the attribute in the business event payload that should be used 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.
display-name-attr	The name of the attribute in the business event payload that should be used 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.

```
<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.

   ```
   <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 `adf-config.xml`. You must add this entry to all `adf-config.xml files`:

```
<wpsC:adf-service-config
xmlns:wpsC="http://xmlns.oracle.com/webcenter/framework/service">
    <resource-handler
class="oracle.apps.fnd.applcore.tags.handler.FndResourceActionViewHandler"/
>
</wpsC:adf-service-config>
```

14.14.6 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 via replies to comments and comments upon comments. This feature in Activity Stream allows users to comment on 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 board, activity stream, doclib and replies on topics discussion forum. In ActivityStream, this feature allows users to indicate if they like a particular Activity.

To enable Comments and Likes for a service, add these ActivityTypes to the `service-definition xml`:

```
<activity-type name="postComment"
               messageFormatKey="ACTIVITY_COMMENT_STATUS"/>
<activity-type name="expressLike"
               messageFormatKey="ACTIVITY_LIKE_STATUS"/>
```

Make sure 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 ActivityMessage.

14.14.7 How to Implement Follow for an Object

Users will be able to see ActivityMessages belonging to the Business Objects they are following. Users should either explicitly follow a business Object, or product teams should provide a way for users to follow certain Business Objects implicitly. A Business Object can be followed for a user by using the WebCenter Follow API. A sample implementation of Follow is shown in Example 14–54.

Example 14–54 Sample Implementation of the Follow Model

```
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);

   ActivityStreamingService asService = ActivityStreamingServiceFactory
     .getInstance().getActivityStreamingService();
   FollowManager followManager = asService.getFollowManager();
   String serviceID = "oracle.apps.fnd.applcore.crmdemo.model.business.CasesEO";
   String userGUID = ApplSessionUtil.getSession().getUserGuid();
   ActivityActor actor = asService.createActor(userGUID);
   String objectTypeName = "Id";
   ServiceObjectType objectType = asService.findObjectType(serviceID,
objectTypeName);
   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");
   }
}
```

14.14.7.1 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 14–55 and Example 14–56.

Note that the id in the service-category-definition file matches the category-id in `service-definition.xml` and it contains "business".

Example 14–55 Sample service-category-definition.xml

```
<service-category-definition xmlns="http://xmlns.oracle.com/webcenter">
 <category id="oracle.apps.fnd.applcore.crmdemo.model.business.CasesEO"

resourceBundle="oracle.apps.fnd.applcore.crmdemo.BusinessActivityServiceResourceBu
ndle"
           titleKey="CASE_SERVICE_CATEGORY"
           icon="/a/s/g.gif"/>
</service-category-definition>
```

Example 14–56 Sample service-category-definition Reference

```
<category-id>oracle.apps.fnd.applcore.crmdemo.model.business.CasesEO</category-id>
```

14.14.7.2 Adding ActivityTypes for Follow and Unfollow

The ActivityTypes shown in Example 14–57 should be added to the service-definitions of all services that use the Follow model. These ActivityTypes are used to construct the message published when an object belonging to the service is Followed or Unfollowed.

Example 14–57 Adding ActivityTypes for Follow and Unfollow

```
<activity-type name="followObject"
   displayName="Follow Object"
   messageFormatKey="ACTIVITY_FOLLOW_OBJECT_MSG"
   description="Follow Object">
</activity-type>

<activity-type name="unfollowObject"
   displayName="Unfollow Object"
   messageFormatKey="ACTIVITY_UNFOLLOW_OBJECT_MSG"
   description="Unfollow Object">
</activity-type>
```

14.14.8 How to Render Contextual Actions in Activity Streams

Contextual Actions are rendered for Business Objects or other resources referenced in ActivityStream messages when contextInfoPopupId is configured in the service-definition.xml file of the Business Object or resource. ActivityStream launches an 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 ActivityStream is used. This is already a requirement for all pages where Contextual Actions-enabled objects are rendered. Activity Stream will make the serviceId, resourceId, and resourceType properties available to the launched pop-up. The pop-up 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 service-definition element, is used to configure the Contextual Actions popup id in service-definition.xml.

```
<contextInfoPopupId>:pt1:r1:casePopup</contextInfoPopupId>
```

The popup sample shown in Example 14–58 uses the serviceId, resourceId, and resourceType properties from ActivityStream that are made available through the launch variable, and makes them available to the popup.

Example 14–58 Sample Popup for ActivityStream

```
<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}"
                       type="popupFetch"/>
</af:popup>
```

14.15 Implementing the Oracle Fusion Applications Search Results UI

The Oracle Fusion Applications Search Results UI provides a front-end, launched from a page using the UI Shell template, that is used to query the Oracle Enterprise Crawl and Search Framework (ECSF).

The minimum requirement is to implement and run a UI Shell Template page. A page using the UI Shell template will automatically 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 will need to make sure that you have followed all of the instructions in Chapter 2, "Setting Up Your Development Environment," Chapter 26, "Getting Started with Oracle Enterprise Crawl and Search Framework," and Chapter 27, "Creating Searchable Objects" and in particular:

- "How to Create a Database Connection" in Chapter 2, "Setting 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.
- Chapter 27, "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 searches go to the Oracle database and are persisted across sessions.

Data Security Integration

Data security, that is, limiting search results to only those items to which the user has authorized access, is handled by the ECSF.

14.15.1 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:

- 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 navigates to a web application that does have the ECSF libraries available.
- Setup switch via JVM system property.

  ```
  -DFUSION_APPS_SEARCH_ENGINE_AVAILABLE=N
  ```

- Use Customization by setting `rendered` to false on the `panelGroupLayout` with id "_UISpg6" (the panel containing the Oracle Fusion Applications Search fields) in the UI Shell main area. Note that by customizing the fields out of the current page, you are not disabling search; the Expression Language bindings on the fields are still evaluated and if the user navigates to a non-customized page, Oracle Fusion Applications Search will be available.
- Setting the profile option **Fusion Apps Search Enabled** to 'N', either at Site or User level.

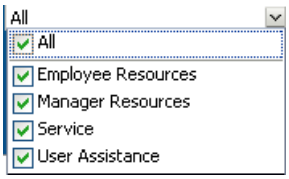
14.15.2 How to Use Basic Search

From the main page of the project in the global area, the Categories and Search terms fields can be seen, as shown in Figure 14–50.

Figure 14–50 Basic Search Fields in UI Shell

If you expand the Categories field, a list, similar to that shown in Figure 14–51, displays:

Figure 14–51 Search Categories Field Expanded



Categories

The end-user can select from the list of Categories and enter a search string. Unchecking the **All** category unchecks all of the categories. The subset of selected categories will be displayed in the entry area of the drop list as a concatenated list separated by a semi-colon (;).

> **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.

Search Term

This is a text field for the values on which to search. The field defaults to showing 20 characters, and can hold a maximum of 2048 characters.

The term is searched in any of the crawled data, which includes the title, fixed and variable content, attached documents, and tags. So if the search term was **foo**, the search returns any data containing the word **foo**.

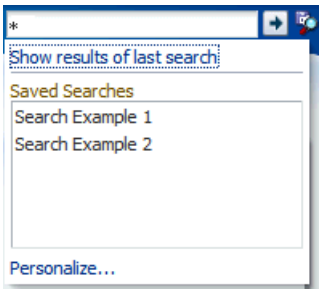
To find *only* items with the *tag* **foo**, enter **ecsf_tags:foo** as the search word. No data will be returned if the word **foo** is in the transactional data but not in the tag.

Click the **Play** button to initiate the search.

Saved Searches

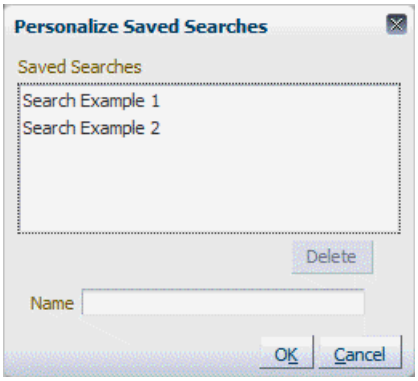
Click the icon to open a list of saved searches. The list, shown in Figure 14–52, includes a **Personalize…** action item that will display the Personalize Saved Searches dialog so that saved searches can be deleted or renamed.

Figure 14–52 Saved Searches List



- **Show Results of Last Search:** Displays the output of the last search.
- **Personalize:** This becomes active if there is a saved search. Click this link to rename or delete a saved search, as shown in Figure 14–53.

Figure 14–53 Personalized Saved Searches Dialog



To rename a saved search, select it, enter a new name in the **Name** field, and click **OK**.

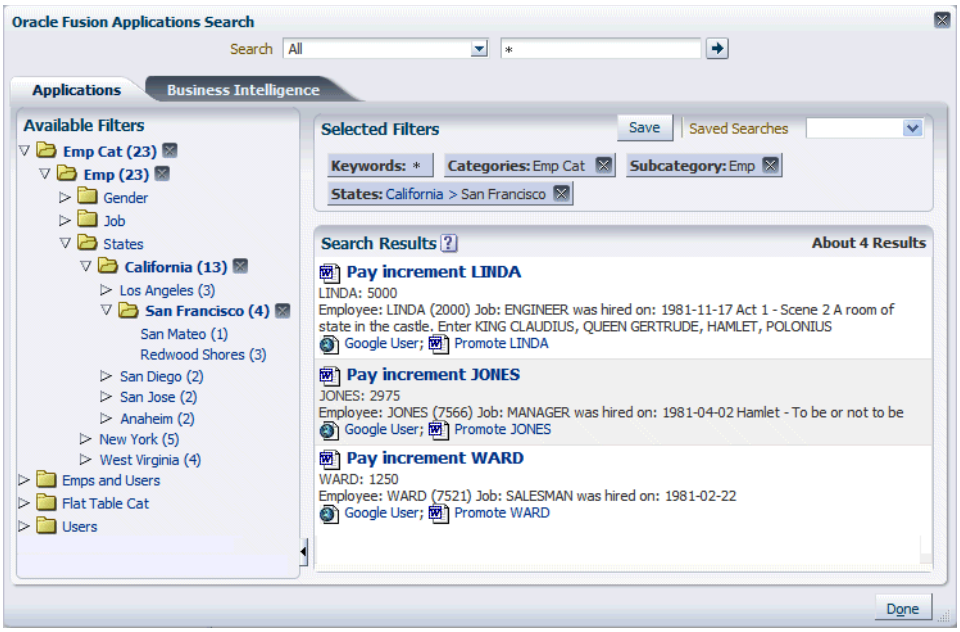
To delete a saved search, select it and click **Delete**.

14.15.2.1 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 14–54 shows typical results.

Note: If a search application, such as Finance or HCM, is down and does not respond to the search request within a pre-determined period of time, the search results will display but there will be a notice that one or more applications did not respond.

Figure 14–54 Search Results Example



To improve performance, the result set size is limited to 10.

The Search Results display consists of:

- A repetition of the fields displayed in the global area.
- A Filter Tree of Categories: Selected filter values will be applied to the search results. A remove filter icon will appear next to a filter value that has been added. Clicking this icon will remove the filter from the search criteria.

  A category is a group of related objects. Examples include any Oracle Fusion business object, and WebCenter objects such as wikis and blogs.

  Searchable Object is the second level. A searchable object is the view object.

  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 14–54, where a State facet contains a City facet, which contains a County facet. Only the name of the highest level facet in the hierarchy is shown.
- A Results section. On the initial query, all selected Categories *that have a non-zero count* will be displayed. That is, if a category has zero results, it will not be displayed. This helps reduce clutter. Only the first category will be displayed expanded. The other Categories will need to be manually expanded.

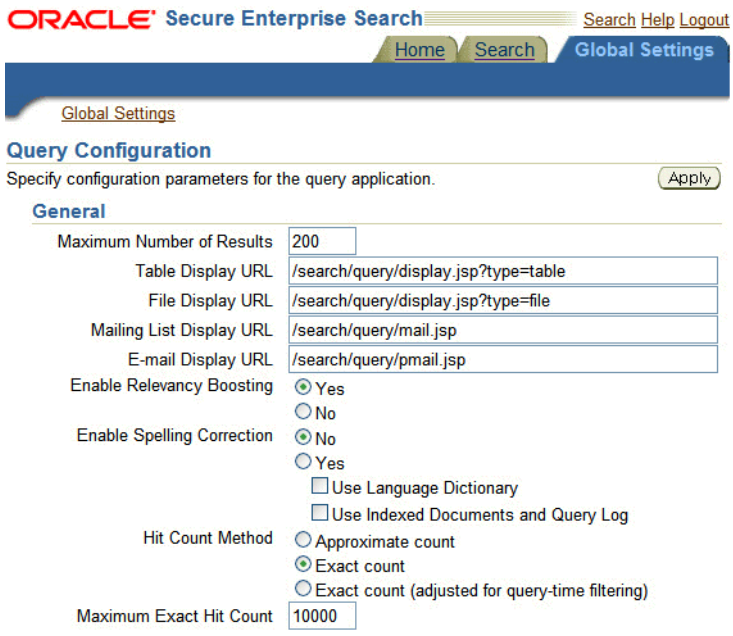
  Each result found under a category provides a navigation link back to the record.

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, launch SES and select **Global Settings > Query Configuration**, and click the **Exact count** radio button, as shown in Figure 14–55. Note that SES warns

against this for performance reasons. See the Oracle Secure Enterprise Search Administration Online Help.

Figure 14–55 Setting the Exact Hit Count in SES



14.15.3 How to Implement the GlobalSearchUtil API

A public API is available to call Oracle Fusion Applications Search without requiring the user to use the global search fields at the top of the UI Shell page template.

You can use this API within your UI, such as in a main area task flow, and bring up the same UI. You must use the UI Shell.

This API, shown in Example 14–59, is available from the `oracle.apps.fnd.applcore.globalSearch.ui` package in the `jdev/oaext/adflib/UIComponents-Viewcontroller.jar` and is the *only* public API supported by Oracle Fusion Applications Search.

Example 14–59 Oracle Fusion Applications Search API

```
/**
 * Run a search from a backing bean and have the search results ui component
 * display.
 * @param searchCategories A list of SearchCategory objects to search within.
 * Can be obtained by calling getCategories() from this class.
 * @param searchString The string to search on
 * @param callerContext a String which represents to the caller, the context
 * in which the search result will be called.  This primarily relates to saved
 * searches, which will be saved with this context, and only saved searches
 * with this context shown to the user.
 * @param e The ActionEvent from the page UIComponent that triggered this
 * functionality.
*/
public static void runSearch(List<SearchCategory> searchCategories,
                             String searchString,
                             String callerContext,
                             ActionEvent e)
```

```
/**
 * Run a search from a backing bean and have the search results ui component
 * display at a set size.
 * @param searchCategories A list of SearchCategory objects to search within.
 * Can be obtained by calling getCategories() from this class.
 * @param searchString The string to search on
 * @param callerContext a String which represents to the caller, the context
 * in which the search result will be called.  This primarily relates to saved
 * searches, which will be saved with this context, and only saved searches
 * with this context shown to the user.
 * @param e The ActionEvent from the page UIComponent that triggered this
 * functionality.
 * @param popupDimension A Dimension object containing the height and width
 * to display the search results popup.
*/
  public static void runSearch(List<SearchCategory> searchCategories,
                               String searchString,
                               String callerContext,
                               ActionEvent e,
                               Dimension popupDimension)
/**
 * Get a list of all the SearchCategory objects.
 * @return  A List of SearchCategory objects containing all possible search
 * categories.
*/
  public static List<SearchCategory> getCategories();
```

14.15.3.1 Using the Search API

Create the component and have an `actionListener` to a backing bean, as shown in Example 14–60. Note that GlobalSearchUtilBean is just an example, not a real bean.

Example 14–60 Creating a Component with actionlistener to Backing Bean

```
<af:commandButton text="commandButton 1"
          actionListener="#{backingBeanScope.GlobalSearchUtilBean.runSearch}">
</af:commandButton>
```

From that backing bean, you can call the Oracle Fusion Applications Search API to run the search, as shown in Example 14–61.

Example 14–61 Calling Oracle Fusion Applications Search API to Run Search

```
public class GlobalSearchUtilBean {
    public GlobalSearchUtilBean()
    {
    }
    public void runSearch(ActionEvent actionEvent)
    {
      List<SearchCategory> categories =GlobalSearchUtil.getCategories();
      // manipulate search category list here
      String searchString = "some search string";
      GlobalSearchUtil.runSearch(categories, searchString, actionEvent);
    }
}
```

14.15.3.2 Running the Oracle Fusion Applications Search UI Under WebLogic Server

For details of running the ECSF artifacts, such as the SearchFeedServlet, see Chapter 27, "Creating Searchable Objects."

To run the UI Shell and Oracle Fusion Applications Search, follow the setup instructions for running Applications Core under WebLogic Server in Chapter 2, "Setting Up Your Development Environment" and the instructions on how to set up a UI Shell page, menu entries and task flows from Section 14.1, "Introduction to Implementing the UI Shell". This should give you a running UI Shell project.

Add the **SearchDB** database connection to the project. For more information about creating the SearchDB connection, see Section 31.6.1, "How to Create the SearchDB Connection on Oracle WebLogic Server Instance".

14.15.4 Introduction to the Crawled Objects Project

The Crawled Objects Project lets you crawl your Search view objects in 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 14–62 shows the UNIX version; the DOS version will be similar.

Example 14–62 Updating ECSF Command Line Script to Reference Applications Core JAR Files

```
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}
```

14.15.5 How to Implement Tags in Oracle Fusion Applications Search

A view object is available to reference Oracle WebCenter Tags. This view object is available in **ORACLE_HOME/jdeveloper/jdev/oaext/adflib/Tags-Model.jar library jar**.

You may use this view object using a view-link and a pre-defined Search Plugin to enable the crawling of Oracle WebCenter Tags, both in initial and incremental (someone has updated the tags) crawls.

Steps:

1. Create your Searchable view object as normal. Example 14–63 uses a Searchable view object over FND_LOOKUPS_VL in the query.

Example 14–63 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
```

The SERVICE_ID specifically identifies your Searchable view object.

The RESOURCE_ID identifies the specific row for the SERVICE_ID. It is essentially a dot-separated primary key of the entity.

These two values will be used when setting up tags in your regular UI and will need to match.

For example, if you have a page with a form and tag button, the Web Center tag would be set up as shown in Example 14–64.

Example 14–64 Setting up the WebCenter Tag

```
<af:panelFormLayout id="pfl1">
   <tag:taggingButton serviceId="oracle.apps.fnd.applcore.lookuptype"
               resourceName="#{bindings.Description.inputValue}"

resourceId="#{bindings.LookupType.inputValue}.#{bindings.ViewApplicationId.inputVa
lue}.#{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>
```

See Section 14.10, "Implementing Tagging Integration" for setting up Tags in your UI. Figure 14–56 shows the attributes for lookup types.

Note: Do not forget to mark your key columns and make sure the order is consistent between the view object and the `tag:taggingButton.resourceId` attribute.

Figure 14–56 Tags - Searchable View Object Lookup Types

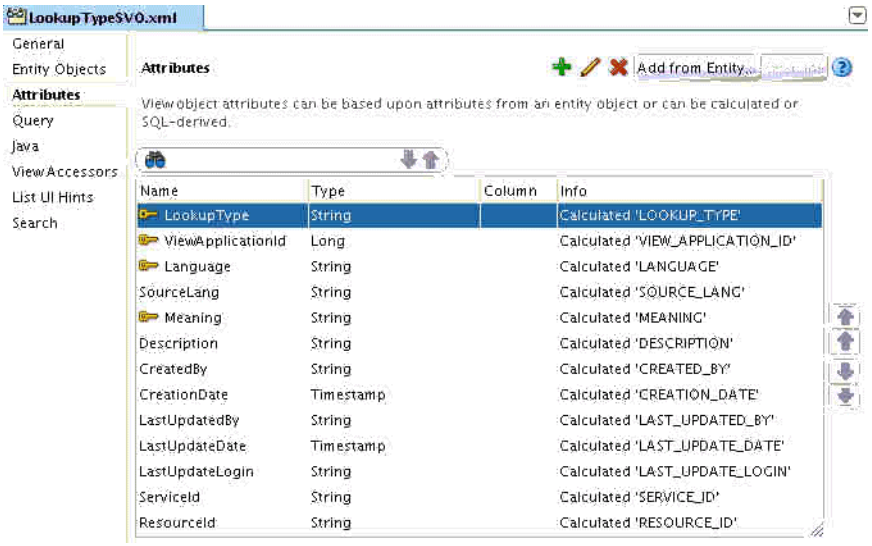
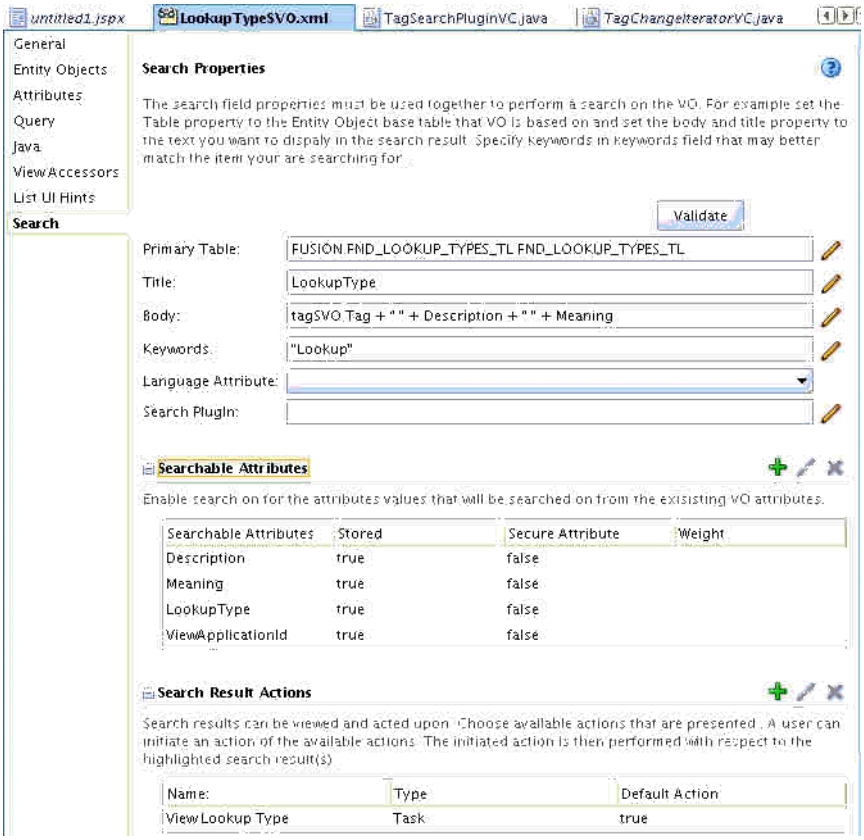


Figure 14–57 shows the attributes for Searchable view object lookup types.

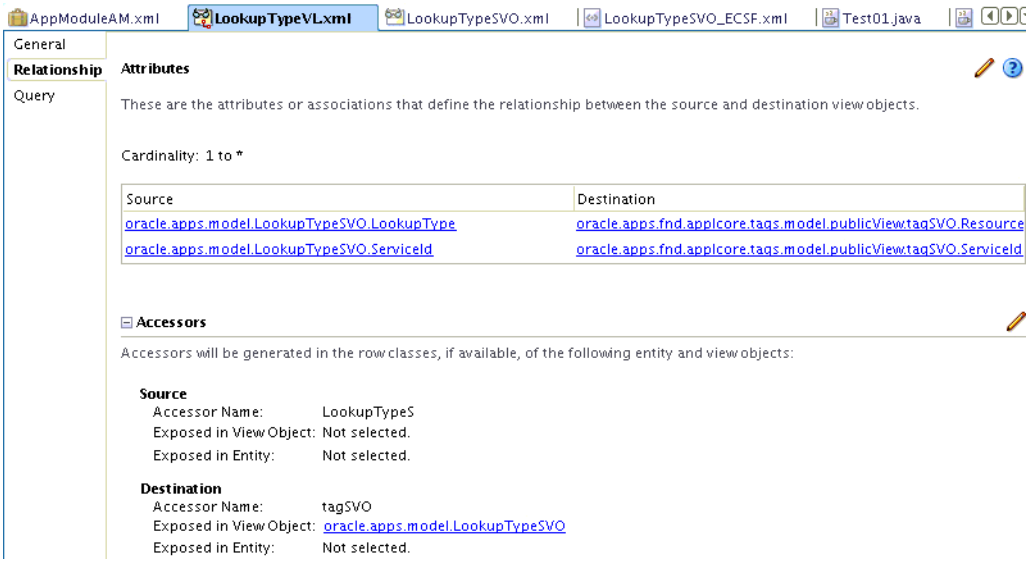
Figure 14–57 Tags - Lookup Types



2. Add a view link to the TagSVO (Service view object) linking the Search view object and Applications Core Tag view object.

   The view link should look similar to Figure 14–58.

Figure 14–58 View Link Example



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 product 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:

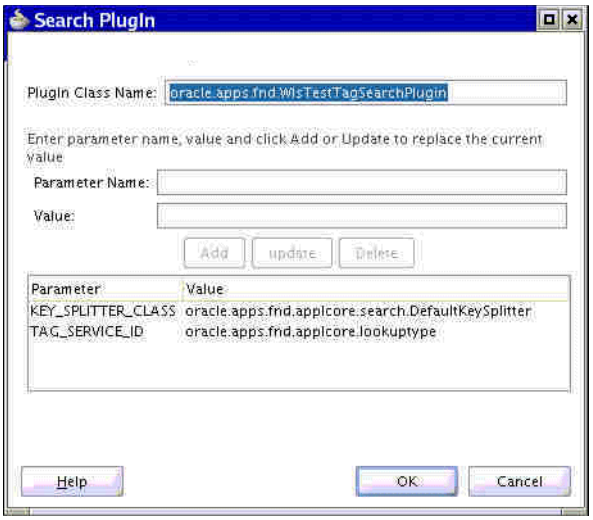
1. Update the Searchable View Object Search Plugin field (see Figure 14–58, "View Link Example") to "oracle.apps.fnd.applcore.search.TagSearchPlugin", or as shown in Example 14–65, create a subclass so that you can incorporate your security rules.

Example 14–65 Creating a Subclass

```
package oracle.apps.fnd;
import oracle.apps.fnd.applcore.search.TagSearchPlugin;
public class WlsTestTagSearchPlugin
extends TagSearchPlugin
{
  // All implementation through super class, or overide 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.
}
```

   Make sure 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. See Figure 14–59.

Figure 14–59 Search Plugin



There are two parameters, shown in Table 14–11, that may be passed to the plugin.

Table 14–11 Parameters that can be passed to the plug-in

Parameter	Required	Description
`TAG_SERVICE_ID`	Yes	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 query.
`KEY_SPLITTER_CLASS`	No	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 PK 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.

2. Start the SearchFeedServlet in the user interface project.

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.

14.15.6 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 Chapter 26, "Getting Started with Oracle Enterprise Crawl and Search Framework."

Figure 14–60 shows the result that will be produced (a single row in the search results table).

Figure 14–60 Search Results Example

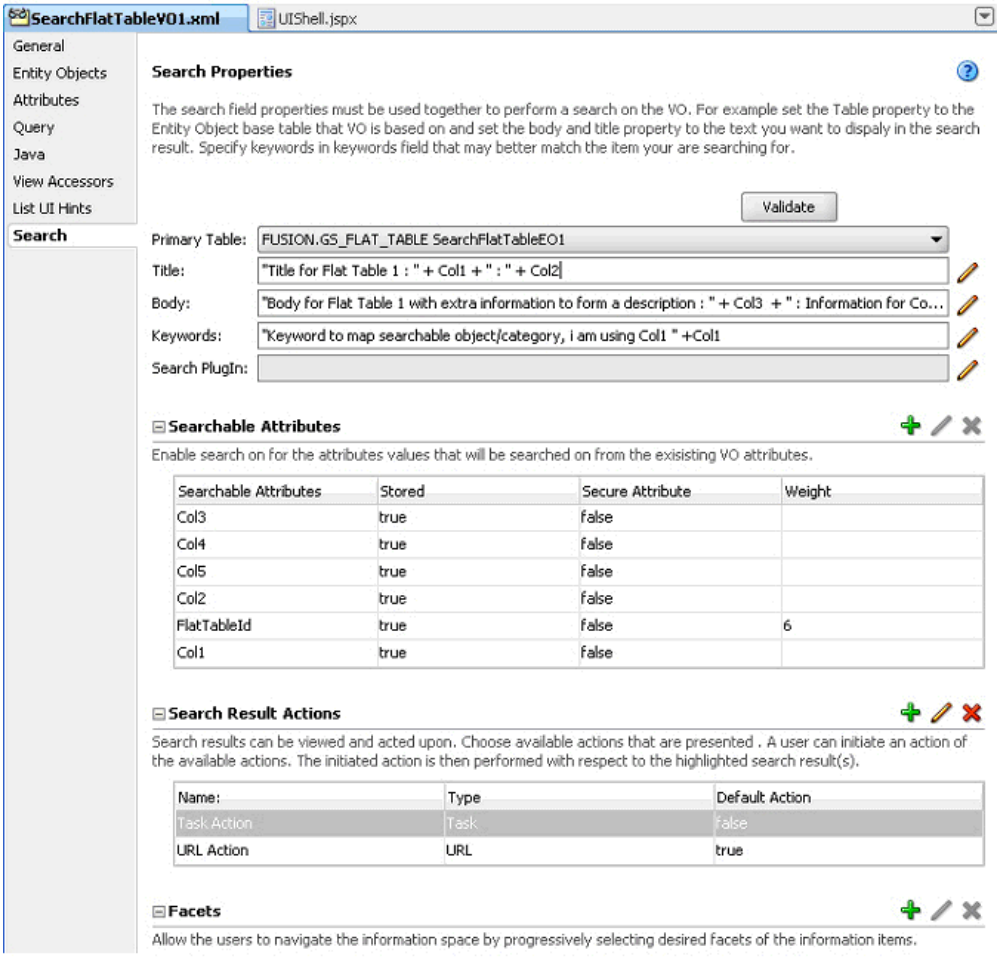
The terminology referred to in this result is:

- **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 14–61, ECSF searchable objects support two distinct Action Types: URL and Task. See also Figure 14–62 and Figure 14–63.

Most Oracle Fusion Applications will use the Task type with specific named parameters to integrate with the UI Shell; however both types will work.

Figure 14–61 Search Properties Example



Fixed Content

The Fixed Content is derived from the Search Properties **Title** field.

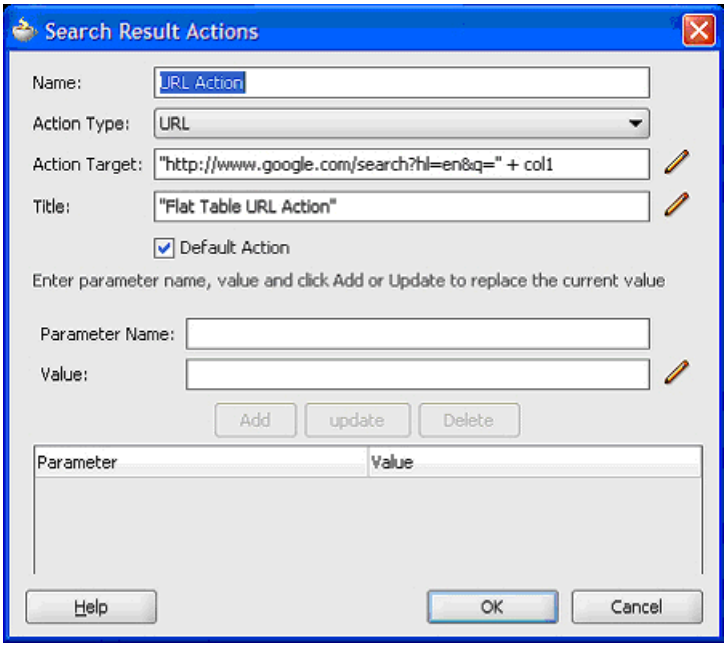
Variable Content

The Variable Content is derived from the Search Properties **Body** field.

14.15.6.1 Implementing the URL Action Type

Figure 14–62 shows a URL Action.

Figure 14–62 Search Result Actions - 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 14–12.

Table 14–12 iconURL Parameter

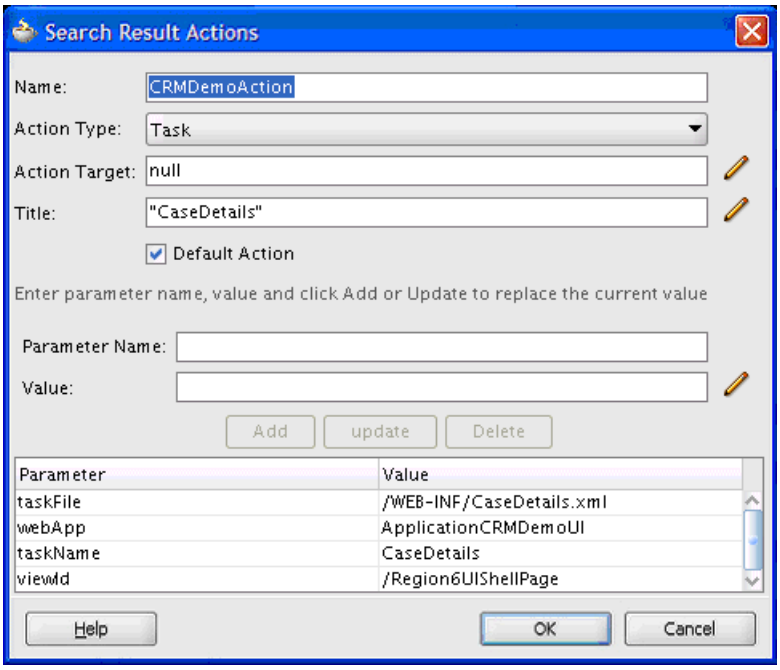
Name	Required	Description
`iconURL`	No	URL of icon to show next to the Action. Can be a relative reference such as /media/search/mime_doc.gif or a full URL such as http://host:port/path/to/icon.gif.

This 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.

14.15.6.2 Implementing the Task Action Type

Figure 14–63 shows a Task Action.

Figure 14–63 Search Result Actions - 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 14–13. Note that, although this table resembles Table 14–14, it presents the use case that the majority of users will use. The information in Table 14–14 is for a very small use case.

  This 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 quotes around the groovy expressions; use single quotes instead.

Table 14–13 Task Action Type Parameters

Name	Required	Description
`viewId`	Y	Name of the page. This is shown in the browser URL bar. For example, in http://127.0.0.1:8989/context-root/faces/TestUIShellPage, it would be "/TestUIShellPage" (Note the leading slash).
`pageParametersList`	N	Parameters list for the page. This is a semicolon delimited String of name value pairs. For example, "param1=value1;param2=value2"
`taskFile`	Y	Name of the task definition file. For example, /WEB-INF/task-flow-definition.xml. See Section 14.15.6.3, "Passing Parameters in Oracle Fusion Applications Search".
`taskName`	Y	The task flow definition id. Available from the task definition file <task-flow-definition> id attribute. For example, <task-flow-definition id='task-flow-definition'> would be "task-flow-definition". See Section 14.15.6.3, "Passing Parameters in Oracle Fusion Applications Search".
`navTaskKeyList`	N	Key list to pass into the task flow to open in the target workspace. This is a semicolon delimited keys or key-value pairs. For example "key1;key2=value2"
`navTaskParametersList`	N	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"
`iconURL`	N	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 'http://host:port/path/to/icon.gif'
`toolTip`	N	Tooltip of action. This also is available for URL actions.
`navTaskLabel`	N	Label to show on the results tab. (Set the tab title of the tab that is opened after clicking a search result action.) If unset, it will use the Action Name (the value shown in the results).
`webApp`	Y	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 Functional Setup tasks.

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 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 (as long as you reference the parameter as a String in the task flow), but for dates and decimals, care should be taken.

14.15.6.2.1 How to Implement Preferred Navigation Fusion Applications 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, 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 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 on caret, and produce an ordered list of navigation targets,
- If there is only one target defined, use it with no permission check.
- For each target, determine if the user can navigate to the page and task flow.
- If a navigable target is in the current view, use it.
- Take the first navigable target.

Whatever the outcome of the permissions check, the developer *must* ensure that at least one target is navigable, 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 14–14. Note that, although this table resembles Table 14–13, it presents a more complicated use case in which teams want to do a security check, and navigate the user to the most secure endpoint (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* only need to use the information in Table 14–13.

Table 14–14 Parameters for Preferred Navigation

Name	Required	Delimited by caret "^"	Description
`applicationStripe`	N	Y	(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
`pageDefinitionName`	N	Y	A delimited String of page definition names. If this parameter is supplied, the applicationStripe parameter must be supplied also. Example: oracle.apps.view.pageDefs.Test1PageDef^oracle.apps.view.pageDefs.AnotherPageDef
`viewId`	Y	Y	Name of the page for the pillar. This is shown in the browser URL bar. For example, in http://127.0.0.1:8989/context-root/faces/TestUIShellPage, it would be "TestUIShellPage^AnotherUIShellPage".
`webApp`	Y	Y	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 Functional Setup tasks.
pageParametersList	N	Y	Parameters list for the page. This is a semicolon delimited String of name value pairs. For example, "param1=value1;param2=value2^anotherParam1=value1;anotherParam2=value2"

Table 14–14 (Cont.) Parameters for Preferred Navigation

Name	Required	Delimited by caret "^"	Description
taskFile	Y	Y	Name of the task definition file. For example, "/WEB-INF/task-flow-definition.xml^/WEB-INF/another task-flow-definition.xml". See Section 14.15.6.3, "Passing Parameters in Oracle Fusion Applications Search".
taskName	Y	Y	The task flow definition id. Available from the task definition file <task-flow-definition> id attribute. For example, <task-flow-definition id='task-flow-definition'> would be "task-flow-definition^another-task-flow-definition". See See Section 14.15.6.3, "Passing Parameters in Oracle Fusion Applications Search".
navTaskKeyList	N	Y	Key list to pass into the task flow to open in the target workspace. This is a semicolon delimited keys or key-value pairs. For example "key1;key2=value2^anotherKey1;anotherKey2=value2"
navTaskParametersList	N	Y	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^anotherParam1=value1;anotherParam2=value2"
navTaskLabel	N	Y	Label to show on the results tab. (Set the tab title of the tab that is opened after clicking a search result action.) If unset, it will use the Action Name (the value shown in the results). For example: "Manage user^View User" (Note: This will be shown on the UI, so use resource bundles.)
iconURL	N	N	URL of the 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 'http://host:port/path/to/icon.gif'.
Tooltip	N	N	Tooltip of action. This also is available for URL actions.

14.15.6.3 Passing Parameters in Oracle Fusion Applications Search

Normally, `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 14–63. The Oracle Fusion Applications Search code will merge them, adding the "#," so they become <taskFile>#<taskName>.

Parameters are always passed as Parameter Name=value. Often, it is either a literal value or an expression such as #{pageFlowScope.val}. Example 14–66 shows how it is done, passing four parameters.

Example 14–66 Parameter Passing in Oracle Fusion Applications Search

```
<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>
      <Parameter Name="navTaskKeyList">
        <Value>
          <![CDATA["meaning=" + Meaning]]>
        </Value>
      </Parameter>
      <Parameter Name="TaskName">
        <Value>
          <![CDATA["LookupTypeSearchResultsTaskFlow"]]>
        </Value>
      </Parameter>
      <Parameter Name="viewId">
        <Value>
          <![CDATA["TestUIShellPage"]]>
        </Value>
      </Parameter>
    </Parameters>
  </Action>
```

14.15.6.4 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 as 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.

14.15.6.5 Using Click Path and the Saved Search

When the user is using Oracle Fusion Applications Search prior to saving a search, he or she may perform a number of 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 is 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 only facet entries 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/expand the attribute filters, the search filters will be filled-in based on this new click path.

14.15.7 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.

14.15.7.1 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 multi-tab format.

The tabs are named Applications, where all Search view objects and Oracle WebCenter results will reside, and 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 lower-case category_name, such as **bi_%**.

Although the formatting of Oracle Business Intelligence results will be slightly different, no developer uptake is required.

Oracle Secure Enterprise Search (SES) Setup

- **Source**

  See the *Oracle Fusion Middleware Developer's Guide for Oracle Business Intelligence Enterprise Edition* 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.BISearchableTreeOb
ject/ConfigFeed?forceInitialCrawl=true
User ID: Administrator
Password: Admin123

Authorization Tab:
HTTP endpoint for authorization:
```

```
http://10.156.30.40:9704/bisearch/crawler/SecurityService
User ID: Administrator
Password: Admin123
Business Component: oracle.biee.search.BISearchableTreeObject
Display URL Prefix: http://10.156.30.40:9704/bisearch/urlbuilder

where the IP address is your Oracle Business Intelligence server installation,
and the username/password are for a sufficiently authorized Oracle Business
Intelligence user.

Leave all other values at default.
```

- **Source Group**

  Create a source group (SES Searchtab > Source groups) and name it **bi_<some code name>**.

  It *must* start with **bi_** so Oracle Fusion Applications Search can recognize it as an Oracle Business Intelligence category.

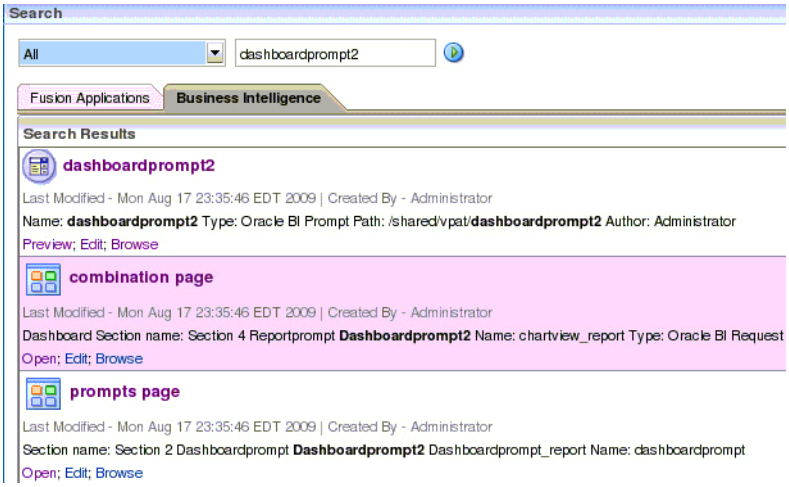
  You may go into Global Settings and translate the group name so the users see a more friendly name.

  Import the group as an external category into ECSF. See "Importing Source Group into ECSF".

- **Searching**

  When Searching, the Oracle Business Intelligence results will display in a separate tab, as shown in Figure 14–64.

Figure 14–64 Oracle Business Intelligence Search Results in New Tab



If there are only Oracle Business Intelligence, or only Oracle Fusion Applications/Oracle WebCenter categories selected, only the one tab appropriate to those categories displays. Otherwise, you can search both and tab between the results.

When you click a link, you will be redirected to Oracle Business Intelligence. If you do not have a consolidated OID setup, you will be asked to log in again.

14.15.7.2 Integrating the Oracle WebCenter

To set up and crawl an Oracle WebCenter environment, see "Managing the Search Service" in *Oracle Fusion Middleware Administrator's Guide for Oracle WebCenter* and then import the searchable objects into ECSF as external categories.

Importing Source Group into ECSF

The group now should be searchable from within the SES Search UI.

To allow the group to be searchable from Oracle Fusion Applications Search, it needs to be imported via the cmdLineAdmin tool or Oracle Enterprise Manager Fusion Middleware Control. Follow these steps if you use the tool:

1. Launch the cmdLineAdmin tool. Its prompt will display.
2. Issue this command at the prompt:

   `> manage instance 124` (where 124 differs for each developer).

   The prompt will change to show that an instance is being managed.
3. Enter this command:

   ```
   Instance: 124> list external categories
   ```

   This information displays:

   ```
   List of External Categories for Instance with ID 124:
   ------------------------------------------------
   ID              | Name                         |
   ------------------------------------------------
   100000000013878  | bi_SearchableTreeDirectory    |
   100000000013879  | WebCenter                     |
   ```

4. Enter this command:

   ```
   Instance: 124> import external categories
   ```

   This command should return an **Import successful** message.
5. Enter this command:

   ```
   Instance: 124> list external categories
   ```

   This information displays:

   ```
   List of External Categories for Instance with ID 124:
   List of External Categories for Instance with ID 124:
   ------------------------------------------------
   ID              | Name                         |
   ------------------------------------------------
   100000000014896  | bi_SearchableTreeDirectory    |
   100000000014897  | WebCenter Jive Forums         |
   100000000014898  | WebCenter Jive Announcements  |
   100000000014899  | WebCenter                     |
   ```

14.15.7.3 Ensuring Parity of Users

Users must be defined in multiple applications if you do not have a single authentication store.

Ensure you have a user defined that is common across both Oracle Fusion Applications and Oracle WebCenter.

For instance, you can create an fmwadmin user on the Oracle Fusion Applications side to do this by adding to the jazn-data.xml file.

With Oracle Secure Enterprise Search (SES) Authentication pointing to the ECSF SearchFeedServlet, which is using the WebLogic Server container security, this user will be verified by the SES authentication callbacks.

In a true enterprise environment, both the Oracle Fusion Applications web container and Oracle WebCenter would be set up with the same OID.

Using two different authentication stores will mean you get multiple logins when clicking results.

14.16 Introducing the Navigate API

Product teams can create a link in the task flow to navigate to a different UI Shell page. Because navigation can occur across different web applications, a single consistent API performs browser redirect. See Table 14–15. This API is exposed as the **FndUIShellController.navigate** Data Control method.

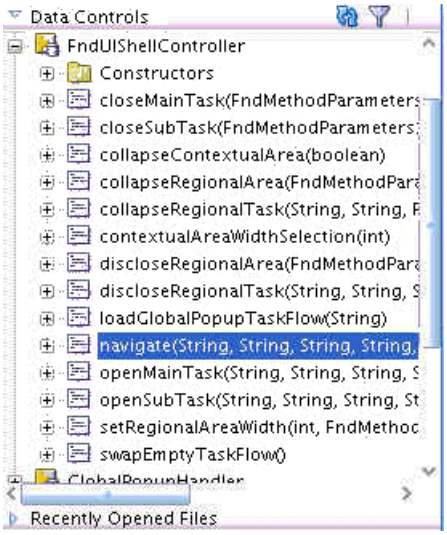
> **Note:** When launching a new window, whenever possible, you should use the Navigate API instead of using the ADF Controller sub-flow calls. This is because the UI Shell has no information about sub-flows. For instance, if you perform a search that returns clickable results and use sub-flows to display 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.

14.16.1 How to Use the Navigate API Data Control Method

Developers will drag and drop the Data Control method on page fragments to create links to invoke navigation.

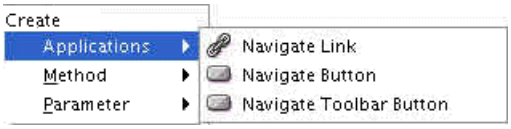
1. Expand the Data Controls and select the `navigate` item, as shown in Figure 14–14.

Figure 14–65 Selecting navigate from Data Controls



2. Drag `navigate` and drop it onto the page fragment. When you do, the Applications Context menu shown in Figure 14–15 displays so you can choose one of the three options.

Figure 14–66 Selecting a Navigate Option from the Applications Context Menu



Developers can specify a task flow to load on the target page. Page level parameters can also be specified.

Table 14–15 Navigate API Parameters

Navigate API Parameter	Attribute Name
`viewId`	navigateViewId
`webApp`	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 Functional Setup Manager tasks. You need to pass the deployed module name to the webApp parameter.
`requestContextPath`	**Obsolete**. Replaced by webApp.
`pageParametersList`	navigateParamsList
`navTaskFlowId`	taskFlowId
`navTaskKeyList`	keyList
`navTaskParametersList`	parametersList Developers 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 "loadDependentFlow=true" to navTaskParametersList. For example: "customerId=123;loadDependentFlow=true"
`navTaskLabel`	label

Certain parameters, summarized in Table 14–16, can be passed using the Navigate API's argument list.

Table 14–16 Parameters Passed Using Navigate API Arguments

Parameter Name	Passed Using	Description
`fndNavForceRefresh`	`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 `fndNavForceRefresh=true` in `pageParametersList` to force the page refresh so that openMainTask would not be used. Example: `pageParametersList = "fndNavForceRefresh=true;param2=value 2";` Details: `pageParametersList` is a semicolon delimited String of name value pairs.
`ContextualAreaCollapsed`	`methodParameters`	Default: false Example: `FndMethodParameters methodParameters = new FndMethodParameters(); methodParameters.setContextualAreaCollapsed( Boolean.TRUE); methodParameters.setContextualAreaWidth(200) ;`
`contextualAreaWidth`	`methodParameters`	Default: 256 Example: `FndMethodParameters methodParameters = new FndMethodParameters(); methodParameters.setContextualAreaCollapsed( Boolean.TRUE); methodParameters.setContextualAreaWidth(200) ;`
`loadDependentFlow`	`navTaskParametersList`	Default: false Example: `navTaskParametersList = "loadDependentFlow=true;param2=value2 ";` Note that the value for `loadDependentFlow` is *case sensitive*. Details: `navTaskParametersList` is a parameter list to pass in to the task flow to open in the target workspace. This is a semicolon delimited string of name/value pairs.

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 Section 14.2.3.4, "Supporting No-Tab Workareas"), 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 14–67.

Example 14–67 Navigating from One Work Space to Another

```
/**
* 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 overriden by passing in
* fndNavForceRefresh=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 needs to 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
*        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.
* @param navTaskFlowId ID of the taskFlow to open in the target workspace
* @param navTaskKeyList Key list to pass into 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."
* @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 void navigate(String viewId, String webApp,
                       String pageParametersList, String navTaskFlowId,
                       String navTaskKeyList,
                       String navTaskParametersList,
                       String navTaskLabel,
                       FndMethodParameters methodParameters)
```

The main navigation is driven by two attributes: `viewId` and `webApp`.

- `webApp` determines which web application to navigate to.
- `viewId` determines the view activity within the target web application.

`pageParametersList` defines custom URL parameters that product teams 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 14–68 shows how to set up these two parameters.

Example 14–68 Setting contextualAreaWidth and contextualAreaCollapsed Parameters

```
FndMethodParameters methodParameters = new FndMethodParameters();
  methodParameters.setContextualAreaCollapsed(Boolean.FALSE);
  methodParameters.setContextualAreaWidth(200);
```

Notes:

- The task flow to be opened in the target Work Area need not 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 ASK deployment tables. These tables are populated at deployment time through Functional Setup 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 cannot be passed as a parameter.
- When navigating using this API, the ADF Controller state of the source page is not cleaned up.

14.16.2 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, product teams 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 run time 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 to which you are trying to navigate.

14.17 Warning of Pending Changes in the UI Shell

When there are pending changes in the UI Shell Main Area, and the user is navigating out of the page, or is refreshing the tab or taskflow, 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 taskflow in MainArea (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 taskflow by using `openMainTask` (with `reuseInstance=true` and either `forceRefresh=true` or with different parameters).
- Relaunching a taskflow using `navigate` (navigating within the same web application and `viewId`).
- Navigating to 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 Application Development Framework. But, from the search results page, drilling down to a subflow and making the flow dirty marks the subflow as a candidate for warn about changes.

14.17.1 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`.

```
<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 14–69.

Example 14–69 Adding the clientListener on a commandButton

```
<af:commandButton actionListener="#{bindings.closeMainTask.execute}"
             text="closeMainTask- with - CL"
```

```
            disabled="false"
            id="cb1">
      <af:clientListener method="queueActionEventOnMainArea" type="action"/>
</af:commandButton>
```

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 14–70.

Example 14–70 Adding a client listener when a command link/button invokes Data Control APIs programmatically

```
<af:commandButton text="Calling datacontrol api programmatically"
         binding="#{backingBeanScope.backing_Navigateviaprogramatically.cb1}"
         id="cb1" action="go">
  <af:clientListener method="queueActionEventOnMainArea" type="action"/>
</af:commandButton>
```

In addition to adding the `clientListener`, when Data Control APIs are executed from within the MainArea, developers must add the `methodAction` shown in Example 14–71 to their main page fragment's pageDef whose taskflow is attached to the tab in MainArea.

Example 14–71 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"/>
```

The Data Control API shown in Example 14–71 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`.

14.17.2 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 14–72.

Example 14–72 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.applcore.patterns.uishell.ui.bean.FndMethodParameters"/>
</methodAction>
```

14.18 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.

Terms

Home page: any one of these JSPX pages.

Oracle Fusion Home: the overall Oracle Fusion Home Page UI.

14.18.1 Supported Behavior

The following key requirements are supported by the Oracle Fusion UI Shell:

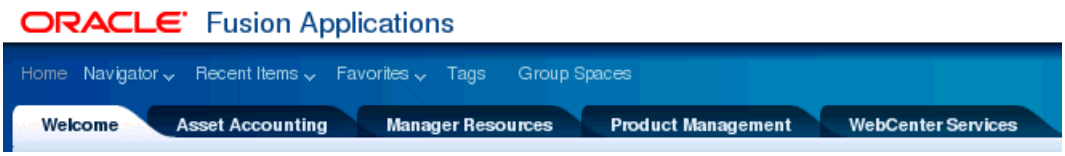
- To conform to Oracle Fusion Applications modularity requirements, the Oracle Fusion Home provides a common entry point across multiple J2EE web applications. A given home page can be hosted on any one of these distinct J2EE web applications. A tab click on a given home page therefore needs to issue a request for another home page that may be hosted on a different web app. 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 link in the Global Area will be disabled on all home pages.
- By default, the Home link will go to the Welcome tab and, if a different tab is selected, the Home link 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 link 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.

14.18.2 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 Steps1 through 4 for all home page JSPX pages.
6. Create the Home Page menu. See Section 14.5, "Working with the Global Menu Model."
7. When running a home page JSPX, the page should look similar to Figure 14–67.

Figure 14–67 Home Page Example



14.18.3 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 14–73.

Example 14–73 Example Use of getURL

```
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
```

- **getURLForCurrentTask**: getURLForCurrentTask, shown in Example 14–74, 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 14–74 Example Use of getURLForCurrentTask

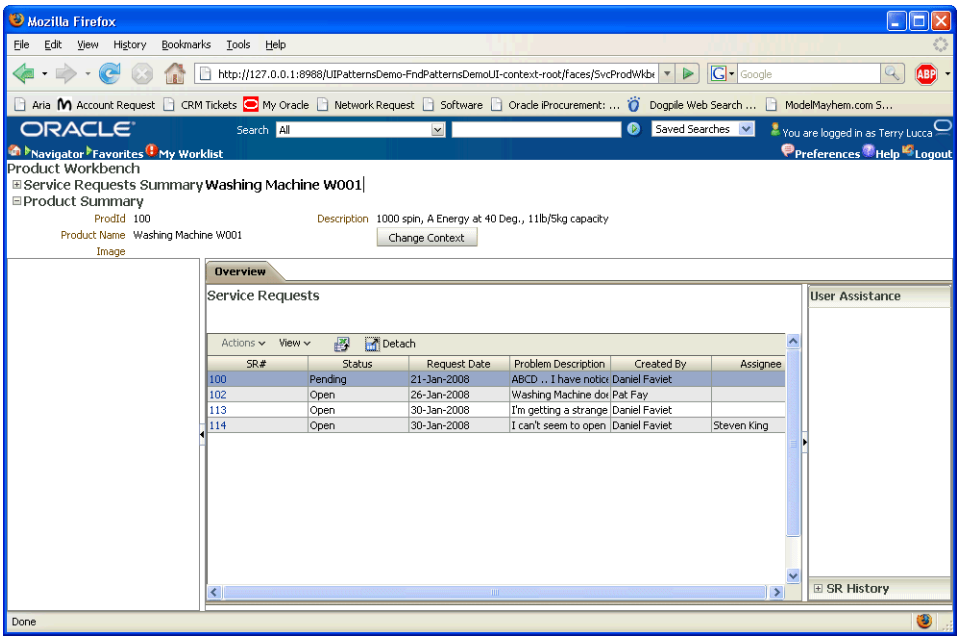
```
public java.lang.String getURLForCurrentTask( )
```

```
throws java.io.IOException
```

14.19 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 14–68.

Figure 14–68 Context Area Example



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.

14.19.1 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 14–75, has been added to adfc-config.xml for viewScope Hashmap.

Example 14–75 Managed Bean viewScope Hashmap Entry in adfc-config.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>
```

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 Chapter 15, "Working with Task Flow Activities," of the *Oracle Fusion Middleware Fusion Developer's Guide for 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.

14.19.1.1 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 Chapter 15, "Working with Task Flow Activities," of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Example 14–76 shows a sample entry in adfc-config.xml for such a view activity.

Example 14–76 Sample Entry in adfc-config.xml for a View Activity

```
<view id="SingleDeptContextPage">
 <page>/oracle/apps/empdeptdemo/ui/page/SingleDeptContextPage.jspx</page>
 <bookmark>
  <url-parameter>
   <name>Deptno</name>
   <value>#{viewScope.fndPageParams.Deptno}</value>
  </url-parameter>
 </bookmark>
</view>
```

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 14–77 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.

Example 14–77 Sample Page Definition for Passing viewScope Values into Context Area Task Flow

```
<taskFlow id="ContextDeptSummary1"

taskFlowId="/WEB-INF/oracle/apps/empdeptdemo/ui/flow/ContextDeptSummary.xml#Contex
tDeptSummary"
              xmlns="http://xmlns.oracle.com/adf/controller/binding">
      <parameters>
        <parameter id="Deptno" value="#{viewScope.fndPageParams['Deptno']}"
                   xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
      </parameters>
    </taskFlow>
```

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 and main area, can access this in pageFlowScope (of main/regional area container task flow owned by Applications Core) for the appropriate input parameters through the menu model and openMainTask API.

Example 14–78 shows a sample entry for a child item node for defaultMain task in the menu.xml to pass the appropriate deptno in the single object context to a task flow that is initialized based on this input value.

Example 14–78 Sample Child Item Node for defaultMain Task

```
<itemNode id="SingleDeptContextPage_EmpListingDefaultTab"
              focusViewId="/SingleDeptContextPage"
              label="#{adfBundle['oracle.apps.empdeptdemo.ui.test_
menuBundle'].EMPLOYEE_LISTING}"
              taskType="defaultMain" reuseInstance="false"

taskFlowId="/WEB-INF/oracle/apps/empdeptdemo/ui/flow/ContainerTF.xml#ContainerTF"
              parametersList="Deptno=#{pageFlowScope.fndPageParams['Deptno']}"/>
```

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.

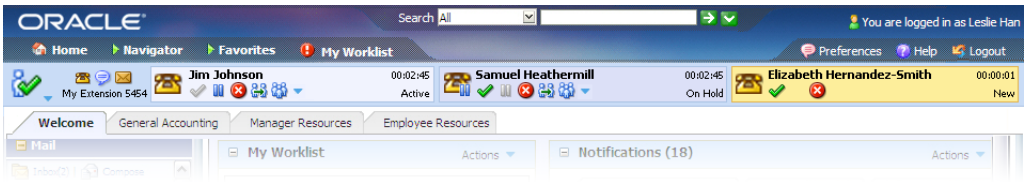
Product teams can also check for the input parameter values within the pageFlowScope of the context area task flow. If invalid, a modal dialog can be launched for the end user to choose a context (by providing links based on the navigate data control method provided by the UI Shell by passing required 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.

14.20 Implementing the Third Party Component Area

The Third Party Component Area, shown in Figure 14–69, 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 14–69 Third Party Component Area Facet Containing A CTI



14.20.1 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 ref 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.

14.21 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:

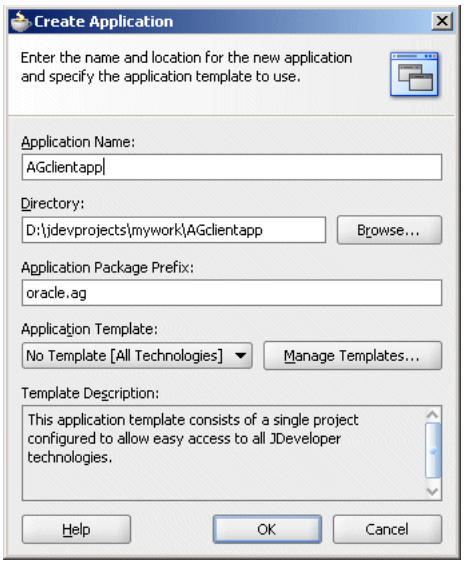
Copy the file `oracle.bpm.activityguide-ui_11.1.1.jar` to a local directory from the following location in the `bpm-jdev-extension.zip` file:

jdev_install`/jdeveloper/soa/modules/oracle.bpm.activityguide-ui_11.1.1.jar`

To develop an activity guide client application using Oracle UI Shell:

1. Create a new application. In the Application Package Prefix field, enter `oracle.ag`.

Figure 14–70 Filling-in the Application Package Prefix Field



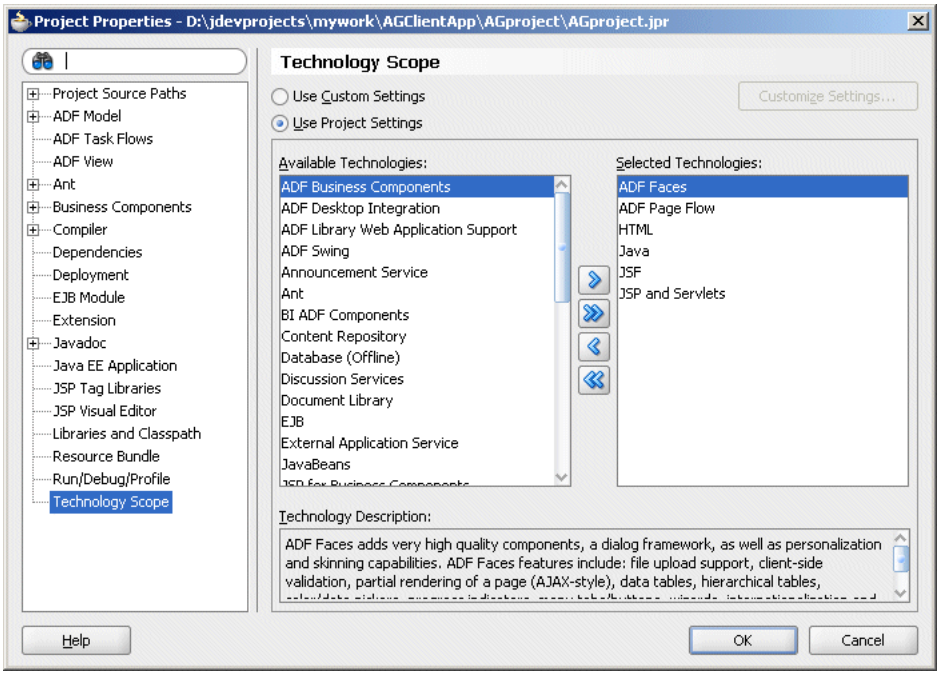
   When prompted to create a new project, click **Cancel**.

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.

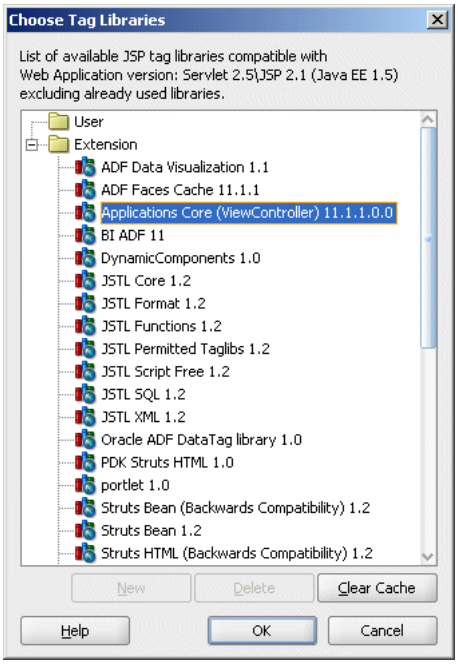
   Make sure the following technologies are also selected: HTML, Java, JSP and JSF and Servlets.

Figure 14–71 Adding the ADF Faces and ADF Page Flow Technologies to the Client Application



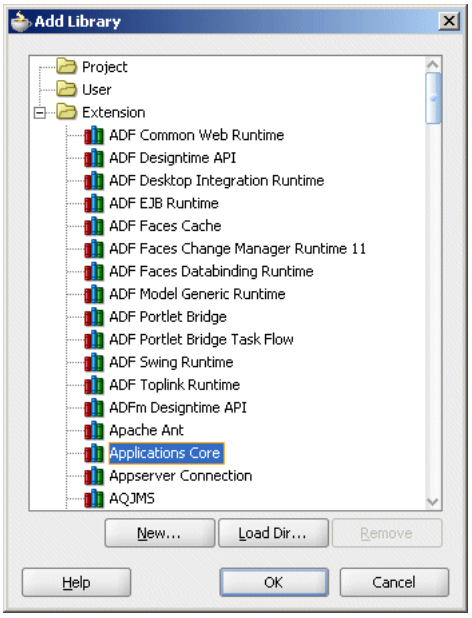
4. Right-click the client application and select **Project Properties** > **JSP Tag Libraries**. Select the **Distributed Libraries** folder and click **Add**. In the Choose Tag Libraries window, select the tag library **Applications Core (ViewController) 11.1.1.0.0** and click **OK**.

Figure 14–72 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 displays, select the **Applications Core** library and click **OK**.

Figure 14–73 Add the Applications Core Library to the Project



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.bpel and oracle.soa.workflow.wc to the `weblogic-application.xml` file.

```
<library-ref>
   <library-name>oracle.soa.bpel</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.bpel_11.1.1/orabpel-common.jar
FMW_home/AS11gR1SOA/soa/modules/oracle.soa.bpel_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 Section 14.3, "Implementing Application Menu Security.".

To add a task flow to the Oracle UI Shell client application:

1. Open the menu metadata menu XML file created in "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 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 displays, 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`. See the table of Activity Guide properties, and the sample properties file, in the "Developing a Guided Business Process Client Application with Oracle ADF" section of *Oracle Fusion Middleware Modeling and Implementation Guide for Oracle Business Process Management*.

   If using identity propagation to secure the Activity Guide, the properties `WorkflowAdminUser` and `WorkflowAdminPassword` are not required.

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 14–79.

Example 14–79 `adfc-config.xml` File with Reference to `activityguide.properties` File

```
<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</value>
   <!-- For example:
   Windows: C:\AG\activityguide.properties
   Linux: /scratch/<user>/AG/activityguide.properties
   -->
</managed-property>
</managed-bean>
```

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 14–80.

Example 14–80 Workflow Services Client Configuration File

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<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</initialContextFactor
y>
          <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>
```

14. Deploy the Acitivity Guide client application with Oracle UI Shell as described in "To deploy an Activity Guide client application with Oracle UI Shell to the integrated Oracle WebLogic Server:" or in "To deploy an Activity Guide client application with Oracle UI Shell to a standalone Oracle WebLogic Server:".

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" chapter of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

To deploy an Activity Guide client application with Oracle UI Shell to the integrated Oracle WebLogic Server:

You can deploy an Activity Guide client application with Oracle UI Shell directly from JDeveloper or to the standalone Oracle WebLogic Server.

From the Application Navigator, right-click the JSF page created in "To develop an activity guide client application using Oracle UI Shell:" and select **Run**.

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 "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.

15

Implementing UIs in JDeveloper with Application Tables, Trees and Tree Tables

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:

- Section 15.1, "Implementing Applications Tables"
- Section 15.2, "Implementing the Applications Tree"
- Section 15.3, "Implementing Applications Tree Tables"
- Section 15.4, "Using the Custom Wizard with Applications Popups"

For basic information, see:

- Chapter 13, "Getting Started with Your Web Interface"
- Chapter 14, "Implementing the UI Shell"
- Chapter 16, "Implementing Applications Panels, Master-Detail, Hover, and Dialog Details"

15.1 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. Once you create an Applications table, you can add table components that allow users to select the table's contents.

Before you begin:

Before you can use Applications tables, you must be familiar with JDeveloper and be able to create JSF pages.

15.1.1 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 15–1 describes Applications table facets and facet contents.

Table 15–1 Applications Table Facets

Facet Name	Description	Values
`table`	Holds ADF table.	ADF Table
`additionalToolbarButtons`	Holds toolbar buttons.	ADF command toolbar buttons under an ADF toolbar
`additionalActionItems`	Holds menu items to be added to default Action pulldown menu items.	ADF menu item component
`appsTableSecondaryToolbar`	Facet for adding more commandToolbar button components to secondary toolbar. (Adds more icons.) 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.	ADF Command Toolbar Buttons
`appsTableStatusbar`	Holds components that contain status bar items. These status bar items are merged with standard items provided by the panelCollection property.	ADF menu item component

Table 15–1 (Cont.) Applications Table Facets

Facet Name	Description	Values
`appsTableViewMenu`	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 `af:group` components containing `af:menuItems`.	ADF menu item component
`appsTableAfterToolbar`	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.	af:toolbar or af:groups of af:toolbars
`popup`	Holds popups. See Section 15.4, "Using the Custom Wizard with Applications Popups." **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: ▪ 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.	Popups under a layout component.

Table 15–2 describes Applications table properties (including properties that are part of the default managed bean), their allowable values, and default actions.

Table 15–2 Applications Table Properties

Property	Description	Values
`id`	Unique identification number for this applications table.	`string`
`rendered`	Whether the applications table is rendered (that is, converted from an object-based description into a graphical image for display).	`boolean expression`
`tableId`	Unique identification number of the underlying ADF table corresponding to this applications table.	`string`

Table 15–2 (Cont.) Applications Table Properties

Property	Description	Values
`createPatternType`	Whether a Create pattern is enabled and, if so, which pattern. User action: Click Create.	`<default>, inline, secondaryWindow, page` `<default>`: No rows are created. You might choose this value if your table is read-only. `inline`: New row is created at the top of the current table. If you choose this value, the create`partialTriggers` property on the ADF table is set automatically. `secondaryWindow`: 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. `page`: An ADFc Controller outcome is returned such that navigation to the next view activity occurs.
`editPatternType`	Whether an Edit pattern is enabled and, if so, which pattern. User action: Click Edit.	`<default>, secondaryWindow, page` `<default>`: No rows become editable. You might choose this value if your table is read-only. `secondaryWindow`: 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. `page`: A standard outcome is returned. In this case, users can edit the values in the currently selected table row.

`duplicatePatternType`	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.
		`<default>, inline, secondaryWindow, page`
		`<default>`: No rows are duplicated.
		`inline`: Selected row is duplicated. If you choose this value, you must also set the partialTriggers property on the ADF table to:
		`<Applications_Table_Id>:duplicate`
		`secondaryWindow`: 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.
		`page`: 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:
		`<Applications_Panel_Id>:<Applications_Table_Id>:duplicate`

`deleteEnabled`	Whether a Delete pattern is enabled. User action: Click Delete.	`boolean` Selected row is deleted.
`createActionListener`	Action binding for the Create button.	`method expression` If defined, this property can be used to supplement the default action specified by the Pattern Type property or completely override it.

Table 15–2 (Cont.) Applications Table Properties

Property	Description	Values
editActionListener	Action binding for the Edit button.	method expression If defined, this property can be used to supplement the default action specified by the Pattern Type property or completely override it.
duplicateActionListener	Action binding for the Duplicate button.	method expression If defined, this property can be used to supplement the default action specified by the Pattern Type property or completely override it.
deleteActionListener	Action binding for the Delete button.	method expression If defined, this property can be used to supplement the default action or completely override it.
createPopupId	Id assigned to the popup to be invoked when users click the Create button.	string
cditPopupId	Id assigned to the popup to be invoked when users click the Edit button.	string
duplicatePopupId	Id assigned to the popup to be invoked when users click the Duplicate button.	string
createText	Value that overrides the default label for Create menu item. It also will be shown as the short description for the Create button.	expression
editText	Value that overrides the default label for Edit menu item. It also will be shown as the short description for the Edit button.	expression
duplicateText	Value that overrides the default label for Duplicate menu item. It also will be shown as the short description for the Duplicate button.	expression
deleteText	Value that overrides the default label for Delete menu item. It also will be shown as the short description for the Delete button.	expression
attachText	Value that overrides the default label for Attach menu item. This attribute is deprecated.	expression
featuresOff	List of default features to turn off for the panelCollection	string
inlineStyle	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.	string

Table 15–2 (Cont.) Applications Table Properties

Property	Description	Values
`styleClass`	styleClass to use for the panelCollection component inside Applications Table component.	string
`exportEnabled`	Rendered attribute for Export button and menu item.	boolean / expression
`createImmediate`	Sets immediate attribute value of "Create" toolbar button and "Create" menu item.	boolean / expression
`deleteImmediate`	Sets immediate attribute value of "Delete" toolbar button and "Delete" menu item.	boolean / expression
`duplicateImmediate`	Sets immediate attribute value of "Duplicate" toolbar button and "Duplicate" menu item.	boolean / expression
`editImmediate`	Sets immediate attribute value of "Delete" toolbar button and "Delete" menu item.	boolean / expression
`attachImmediate`	Sets immediate attribute value of "Attach" toolbar button and "Attach" menu item. **Note:** This attribute is deprecated.	boolean / expression
`primaryToolbarRendered`	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.	boolean / expression
`secondaryToolbarRendered`	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.	boolean / expression

Table 15–2 (Cont.) Applications Table Properties

Property	Description	Values
`<button_ name>PartialTriggers` For example: `deletePartialTriggers`	Partial triggers attribute for the <button_name> toolbar button. 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 commandToolbarButton and the menu item. For example, when the createPartialTriggers is set in the Applications Table, the value for this attribute is set on the partialTrigger property of both the create command toolbar button and create menu item.	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 editDiabled, 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 once 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 or not 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.
`createAction`	Action binding for the Create icon.	method expression
`editAction`	Action binding for the Edit icon.	method expression
`duplicateAction`	Action binding for the Duplicate icon.	method expression
`deleteAction`	Action binding for the Delete icon.	method expression
`createEnabled`	Rendered attribute for create	Boolean value or EL Expression
`duplicateEnabled`	Rendered attribute for duplicate	Boolean value or EL Expression
`editEnabled`	Rendered attribute for edit	Boolean value or EL Expression
`createPartialSubmit`	PartialSubmit attribute for create	Boolean value or EL Expression
`createDisabled`	Disabled attribute for create	Boolean value or EL Expression
`editDisabled`	Disabled attribute for edit	Boolean value or EL Expression

Table 15–2 (Cont.) Applications Table Properties

Property	Description	Values
`duplicateDisabled`	Disabled attribute for duplicate	Boolean value or EL Expression
`deleteDisabled`	Disabled attribute for delete	Boolean value or EL Expression
`confirmDelete`	Set this value if you want delete confirmation to be displayed. The default message is **The selected record(s) will be deleted. Do you want to continue?** To change this, use the `deleteMsg` attribute.	Boolean value or EL Expression
`deleteMsg`	Provide a customized delete confirmation message that can be shown in the popup.	String value or EL Expression
`actionsMenuRendered`	Rendered attribute for Actions menu	Boolean value or EL Expression
`actionsContentDelivery`	ContentDelivery attribute for Actions menu. This attribute can take two values. ■ lazy ■ immediate The default value is **immediate**. Setting the attribute value to **lazy**: ■ Provides better performance. ■ Does not allow for enabling/disabling of the menu items based on client actions, such as current row selected.	String value
`toggleEditRendered`	The toggleEditRendered feature is used to render the editAll or clickToEdit choices for Applications Table. See Section 15.1.2.2.3, "Toggle Click to Edit / Edit All in Applications Table."	Boolean. Default value is **False**.

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.applcore.patterns.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 15–1.

Example 15–1 Defining a Managed Bean Class to Attach a Custom Handler to a Button

```
import oracle.apps.fnd.applcore.patterns.ui.ApplicationsTableEventHandler;
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.
    ApplicationsTableEventHandler appTableHandler =
```

```
ApplicationsTableEventHandler.getInstance();
    String outcome = appTableHandler.processCreate();

    // If popup is required to be invoked after event handling
    PatternUtils.invokePopup(popupId);

    return outcome;
  }
}
```

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.

15.1.2 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. Once the table is created, you can change any parameters from the Property Inspector.

15.1.2.1 Adding 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.

15.1.2.2 Adding 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:

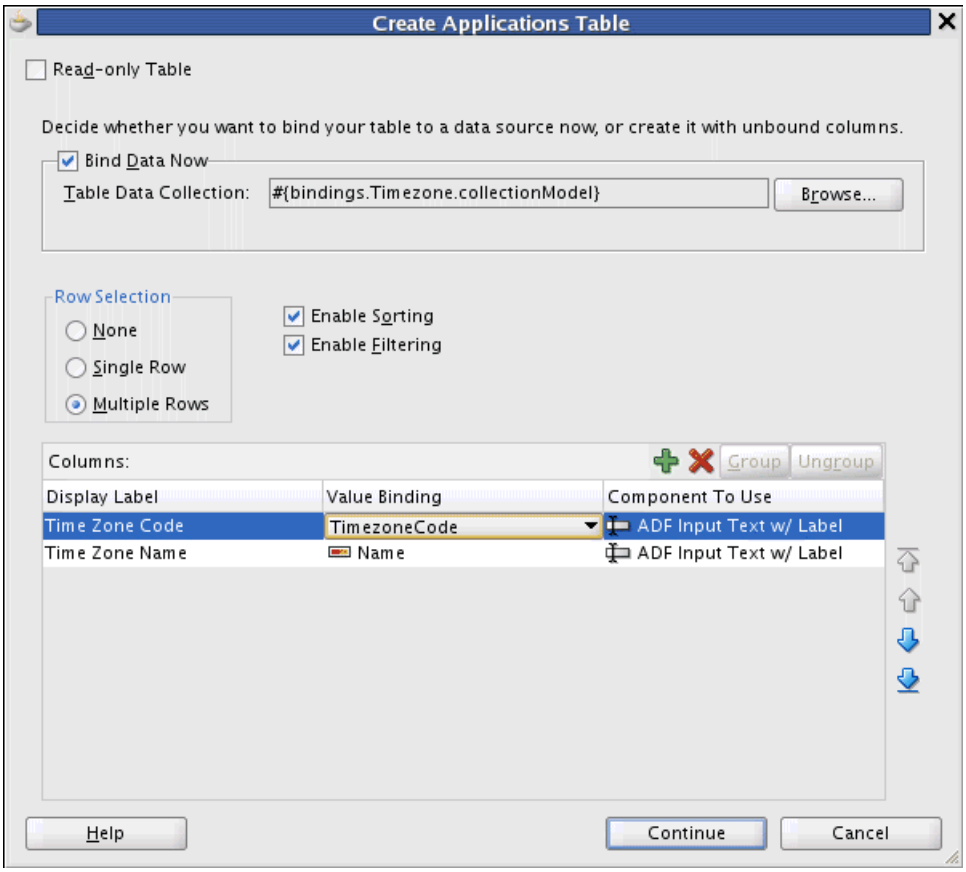
- Bind your table to a data source
- Create placeholder columns and define their attributes
- Enable table ADF behaviors
- Select table default actions

The Applications Table wizard has two dialogs. Click **Cancel** in either dialog to cancel your actions and exit the wizard. Click **Next** to accept the defaults.

To add an Applications Table using the Applications Table wizard:

When the Applications Table wizard is launched, the Create Applications Table dialog is displayed, as shown in Figure 15–1.

Figure 15–1 Create Applications Table Dialog

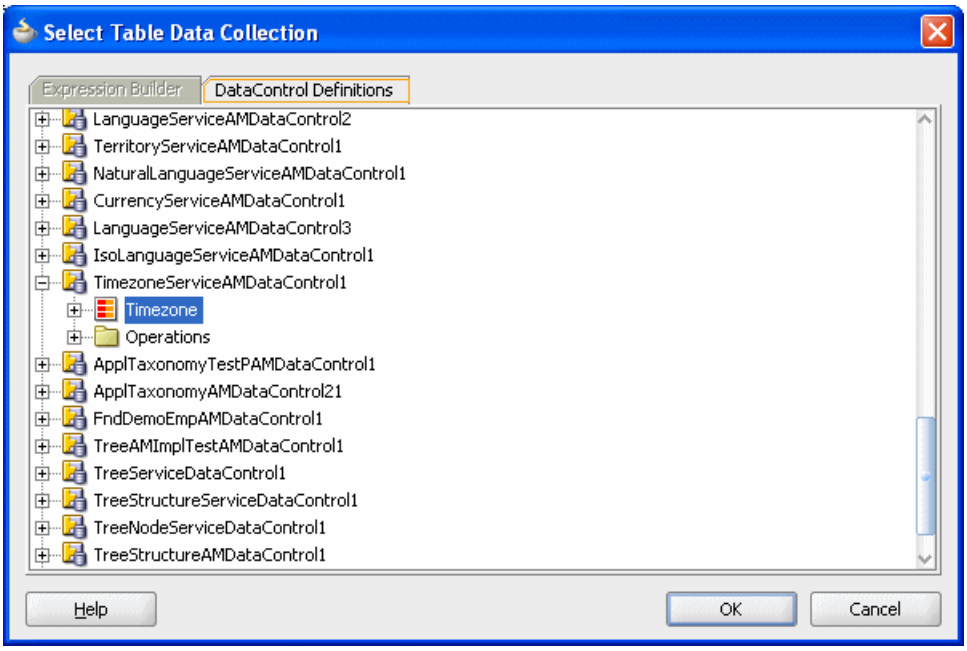


1. In the Create Applications Table dialog:
   a. Select **Read-only Table** to prevent users from modifying the data. If you select Read Only, the options in the Component To Use column will change from Input to Output components.
   b. Bind data to the table (optional):
      – Select the **Bind Data Now** box to bind a data control to the table.
      – In the **Table Data Collection** section, 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, as shown in Figure 15–2.

Figure 15–2 Select Table Data Collection Dialog



– Select the data source to bind to your table and click **OK**.

When you bind the data, the table creates placeholder columns that can be used for layout purposes.

c. 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.

d. 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 EL 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.

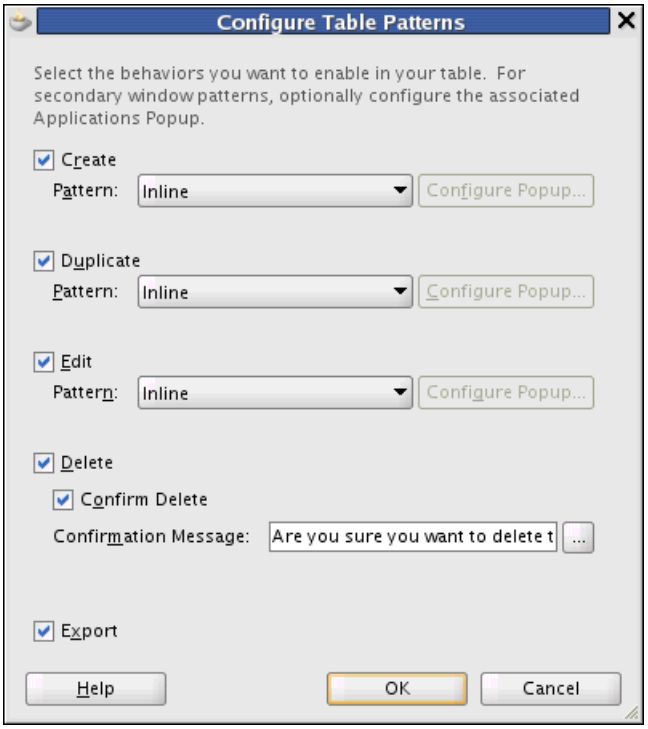
Sort Column Order

Click the up or down arrows to sort the order of the columns.

e. Click **Continue**.

The Configure Table Patterns dialog is displayed, as shown in Figure 15–3.

Figure 15–3 Configure Table Patterns Dialog



2. In the **Configure Table Patterns** dialog, select default actions for your Applications table (optional):
   - **Create / Pattern:** Creates a table row.
     - Once you have chosen to enable row creation, choose a pattern from the list to invoke an action.
     - If you choose the Secondary Window pattern, click **Configure Popup** to display the Applications Popup wizard (see Section 15.4, "Using the Custom Wizard with Applications Popups") and follow the instructions to configure the popup associated with this pattern. See Table 15–1, " Applications Table Facets" for important information about the popup's **Cancel** button.
   - **Duplicate / Pattern:** Duplicates the row.
     - Once you have chosen to enable row duplication, choose a pattern from the list to invoke an action.
     - If you choose the Secondary Window pattern, click **Configure Popup** to display the Applications Popup wizard (see Section 15.4, "Using the Custom Wizard with Applications Popups") and follow the instructions to configure the popup associated with this pattern. See Table 15–1, " Applications Table Facets" for important information about the popup's **Cancel** button.
   - **Edit / Pattern**: Enables row modification.
     - Once you have chosen to enable row editing, choose a pattern from the list to invoke an action.
     - If you choose the Secondary Window pattern, click **Configure Popup** to display the Applications Popup wizard (see Section 15.4, "Using the

Custom Wizard with Applications Popups") and follow the instructions to configure the popup associated with this pattern. See Table 15–1, " Applications Table Facets" for important information about the popup's **Cancel** button.

- **Export**: Export the data to a Microsoft Excel-compatible file.
- **Delete**: Allows users to delete the row.

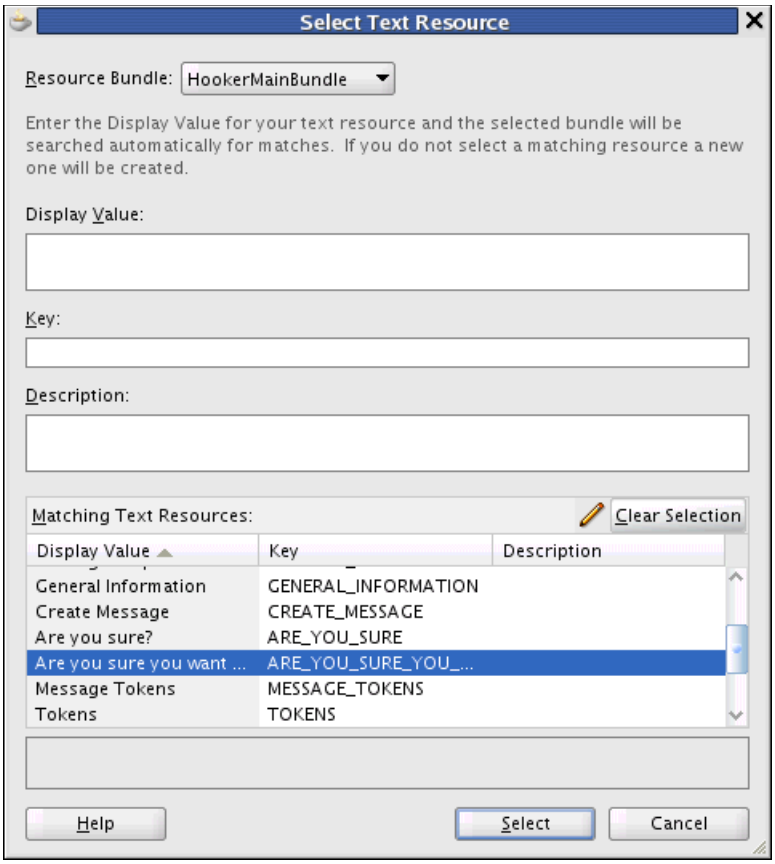
  Confirm Delete: Select this option so that the default **The selected record(s) will be deleted. Do you want to continue?** prompt displays in a popup when the delete row function is used.

  When you set the `confirmDelete` attribute to **true**, the confirmation popup displays and the row is deleted when you click **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 Section 15.1.2.2.1, "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, as shown in Figure 15–4.

Figure 15–4 Selecting an Existing Delete Confirmation Message

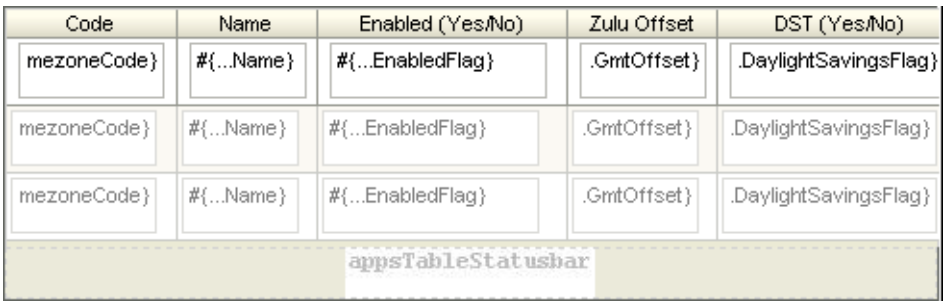


If you need to manually create or edit the Delete Confirmation parameter, see Section 15.1.2.2.1, "Manually Enabling Delete Confirmation."

For more information on pattern types, see Table 15–2, " Applications Table Properties".

3. Click **OK** to save your choices and create the Applications table, or click **Cancel** to delete your choices.
4. If you click **OK**, the table and its components appear in the editor, as shown in Figure 15–5.

Figure 15–5 Applications Table and Its Components in the Editor



15.1.2.2.1 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.

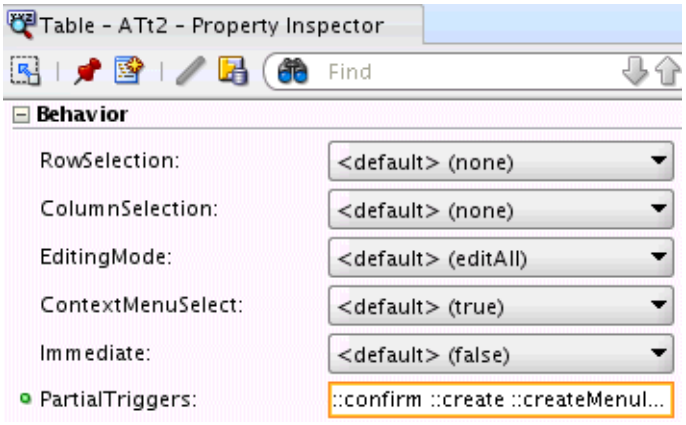
See Example 15–2 for sample code that shows both the delete confirmation enabled and the custom message.

Example 15–2 Sample Code Showing Delete Confirmation and Custom Message

```
<fnd:applicationsTable tableId="ATt2" id="AT2" confirmDelete="true"
        deleteMsg="#{viewcontrollerBundle.ARE_YOU_SURE_YOU_WANT_TO_DELET}"
        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">
```

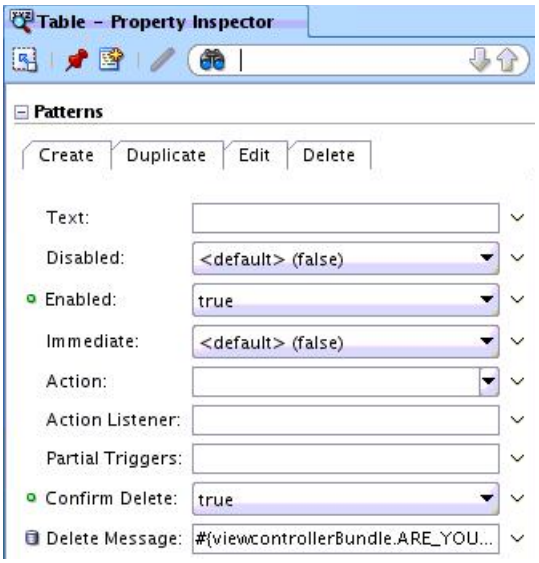
With **af:table** selected in the Structure view, Figure 15–6 shows the PartialTriggers entries in the Property Inspector view.

Figure 15–6 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 15–7.

Figure 15–7 Message Settings in Property Inspector



15.1.2.2.2 Multiple Row Selection on Table If multiple row selection is enabled, editing functions will behave as shown in Table 15–3.

Table 15–3 Function Behavior with Multiple Row Selection Enabled

Function	Behavior
Delete	Selecting more than one row and selecting Delete deletes all the selected rows.
Create	Selecting more than one row and selecting Create will create a new row as the first row.
Edit	Selecting more than one row and selecting Edit will show an alert window asking you to select a single row to edit.
Duplicate	Selecting more than one row and selecting Duplicate will show an alert window asking you to select a single row to duplicate.

Enabling Multiple Row Selection Manually

If multiple row selection is selected in the Create Applications Table wizard (see Figure 15–1), 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 15–3.

Example 15–3 Example of Enabling Multiple Row Selection

```
<fnd:applicationstable tableid="att3" id="at3" deleteenabled="true"
             createpatterntype="inline"
             duplicatepatterntype="inline"
             editpatterntype="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">
```

15.1.2.2.3 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 15–8 shows the Edit All menu option and icon when the table is in the Click to Edit mode.

Figure 15–8 Table Edit All Menu Option and Icon

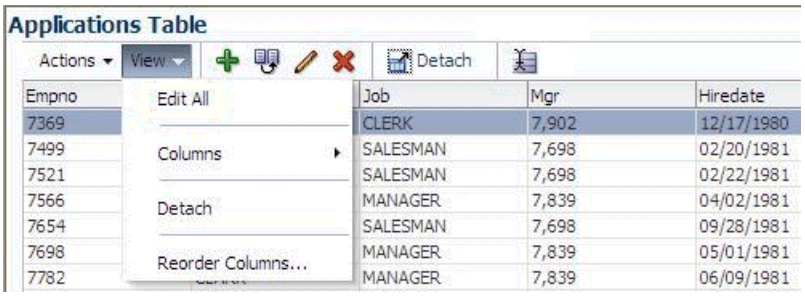
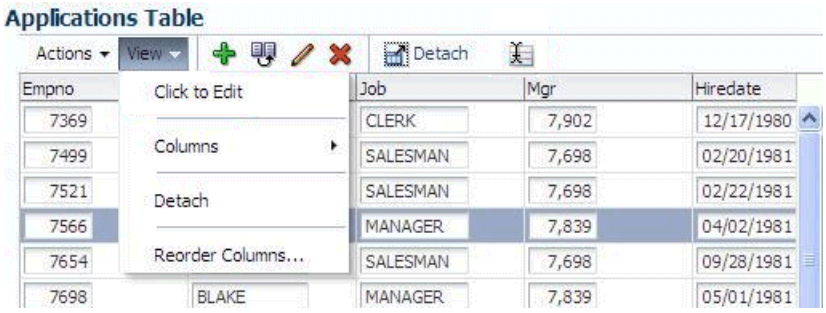


Figure 15–9 shows the Click to Edit menu option and icon when the table is in the Edit All mode.

Figure 15–9 Table Click to Edit Menu Option and Icon



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 15–2) on the Applications Table and does not happen automatically.

15.1.3 Introduction to Selected Elements in the Table Property Inspector

When a table is added to a page, code similar to that shown in Example 15–4 is inserted and displayed in the Source view.

Example 15–4 Sample Code Added When a Table Is Added

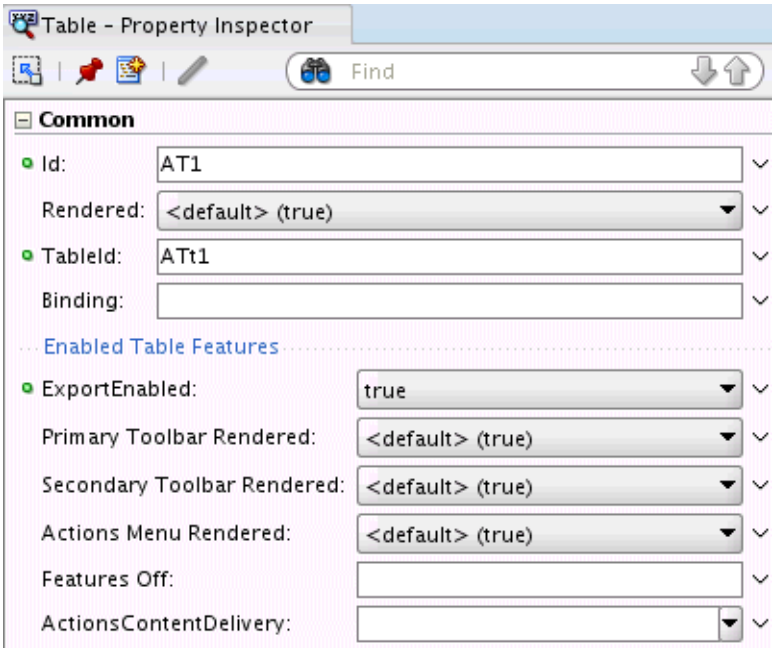
```
<fnd:applicationsTable tableId="ATt1" id="AT1" confirmDelete="true"
            deleteMsg="#{viewcontrollerBundle.DO_YOU_WANT_TO_DELETE_THIS_ROW}"
            deleteEnabled="true" createPatternType="inline"
            duplicatePatternType="secondaryWindow"
            editPatternType="inline" exportEnabled="true"
            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}"
        rowBandingInterval="0"
        selectedRowKeys="#{bindings.ServiceRequests1.collectionModel.selectedRow}"

selectionListener="#{bindings.ServiceRequests1.collectionModel.makeCurrent}"
        rowSelection="single" id="ATt1"
        partialTriggers="::confirm ::create ::createMenuItem">
```

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.

15.1.3.1 Common Properties Section

The Common properties section of the Table Property Inspector resembles Figure 15–10.

Figure 15–10 Common Properties Section



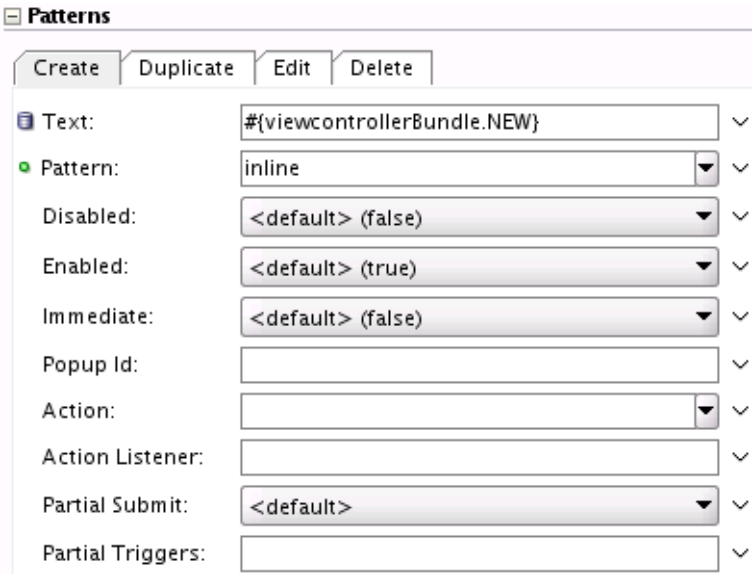
The selected Common settings are:

- **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.

15.1.3.2 Patterns Properties

The Patterns properties section of the Table Property Inspector resembles Figure 15–11. The properties for Create, Duplicate, Edit, and Delete are the same.

Figure 15–11 Patterns Properties Section



The selected Patterns settings are:

- **Disabled:** Sets whether or not 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 or not 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 15–5 sets the partialTrigger attribute value on the Create button icon and Create menu item

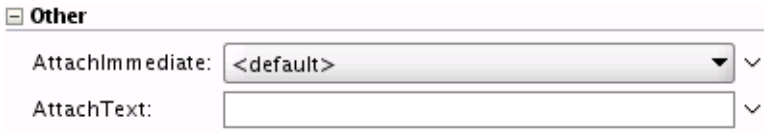
Example 15–5 Example of a Partial Trigger

```
<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="ATt1" id="AT1" deleteEnabled="true"
          createPatternType="inline"
          duplicatePatternType="inline"
          editPatternType="inline"
          createText="#{viewcontrollerBundle.NEW}"
       createDisabled="#{bindings.LedgerId.inputValue == null}"
          createPartialTriggers=":::ledgerIdId">
```

15.1.3.3 Other Properties

The Patterns properties section of the Table Property Inspector resembles Figure 15–12.

Figure 15–12 Other Properties Section



- **AttachImmediate** and **AttachText**: Do not use. These have been deprecated.

15.1.4 How to Modify Applications Table Components and Properties

Once 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.

15.1.4.1 Adding Data Controls to Tables

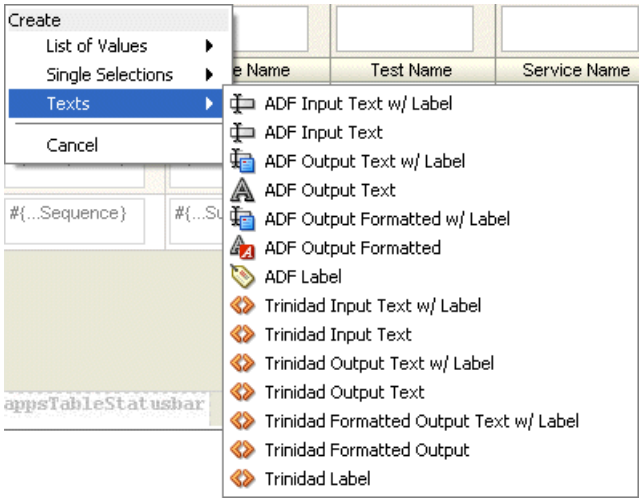
To add Data Controls to tables:

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 15–13.

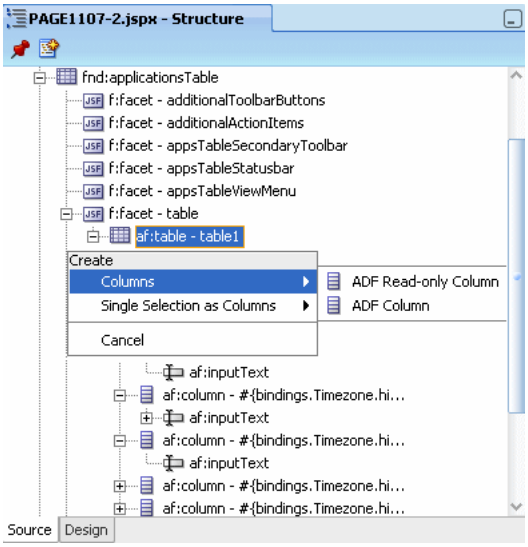
Figure 15–13 Example of Context Menu Choices in Design View



- 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: **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 15–14, 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.

Figure 15–14 Example of Context Menu Choices in Structure View



15.1.4.2 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 15–6:

Example 15–6 Surrounding the Toolbar Buttons with af:toolbar

```
<f:facet name="additionalToolbarButtons">
   <af:toolbar>
     <af:commandToolbarButton text="Button1"/>
     <af:commandToolbarButton text="Button2"/>
   </af:toolbar>
</f:facet>
```

> **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.

15.1.4.3 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 "Organizing Content on Web Pages" and "Customizing the Appearance Using Styles and Skins" in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

15.1.4.4 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"
```

15.1.5 What Happens When You Add an Applications Table

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.

15.2 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

15.2.1 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 15–4 are exposed on the Applications Tree.

Table 15–4 Applications Tree Facets

Facet	Description	Allowed Children
`tree`	Facet for holding the ADF tree	ADF Tree
`additionalToolbarButtons`	Facet for adding toolbar button icons by the developer.	ADF Command Toolbar Buttons
`additionalActionItems`	Facet for adding more menu items to default menu items.	ADF menu item component
`appsTreeSecondaryToolbar`	Facet for adding more commandToolbar components to secondary toolbar.	ADF Command Toolbar Button component
`appsTreeStatusbar`	Facet for adding component containing statusbar item(s). These statusbar items are merged with standard items provided by the panelCollection.	ADF component
`appsTreeViewMenu`	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.	ADF menu item component
`appsTreeAfterToolbar`	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.	"af:toolbar" or "af:groups" of "af:toolbars"

Table 15–4 (Cont.) Applications Tree Facets

Facet	Description	Allowed Children
`popup`	Facet for adding popups. See Section 15.4, "Using the Custom Wizard with Applications Popups." **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.	Any number of popups under a layout component

The properties shown in Table 15–5 are exposed on the Applications Tree.

Table 15–5 Exposed Applications Tree Properties

Property	Description	Allowed Values
`id`	The unique ID for this Applications Tree	string
`rendered`	Whether the Applications Tree is rendered or not	boolean / expression
`treeId`	The unique ID of the ADF tree underneath this Applications Tree	string
`createPatternType`	Whether any Create pattern is enabled, and if yes, which pattern	none, secondaryWindow (then Create Popup Id must also be set), page
`editPatternType`	Whether any Edit pattern is enabled, and if yes, which pattern	none, secondaryWindow (then Edit Popup Id must also be set), page
`duplicatePatternType`	Whether any Duplicate pattern is enabled, and if yes, which pattern	none, inline (see "Inline Duplicate Pattern Type"), secondaryWindow (then Duplicate Popup Id must also be set), page
`createEnabled`	Rendered attribute for create	string
`editEnabled`	Rendered attribute for edit	boolean value or Expression Language expression
`duplicateEnabled`	Rendered attribute for duplicate	boolean value or Expression Language expression
`deleteEnabled`	Rendered attribute for delete	boolean value or Expression Language expression
`createAction`	Action binding for the Create icon	method expression
`editAction`	Action binding for the Edit icon	method expression
`duplicateAction`	Action binding for the Duplicate icon	method expression
`deleteAction`	Action binding for the Delete icon	method expression

Table 15–5 (Cont.) Exposed Applications Tree Properties

Property	Description	Allowed Values
createActionListener	Action listener binding for the Create icon	method expression If defined, this property can be used to supplement the default action specified by the Pattern Type property or completely override it.
editActionListener	Action listener binding for the Edit icon	method expression If defined, this property can be used to supplement the default action specified by the Pattern Type property or completely override it.
duplicateActionListener	Action listener binding for the Duplicate icon	method expression If defined, this property can be used to supplement the default action specified by the Pattern Type property or completely override it.
deleteActionListener	Action listener binding for the Delete icon	method expression If defined, this property can be used to supplement or override the default action.
createPopupId	ID of the popup to be invoked when Create button is clicked	string
editPopupId	ID of the popup to be invoked when Edit button is clicked	string
duplicatePopupId	ID of the popup to be invoked when Duplicate button is clicked	string
createText	Overrides default label for Create menu item. This value will also be shown as the short description for the Create button.	expression
editText	Overrides default label for Edit menu item. This value will also be shown as the short description for the Edit button.	expression
duplicateText	Overrides default label for Duplicate menu item. This value will also be shown as the short description for the Duplicate button.	expression
deleteText	Overrides default label for Delete menu item. This value will also be shown as the short description for the Delete button.	expression
exportEnabled	Whether export is enabled	boolean / expression
featuresOff	A list of default features to turn off for the panelCollection, such as detach (see featuresOff attribute of panelCollection for more details)	string
inlineStyle	The CSS styles to use for the panelCollection component inside the Applications Tree component. This is intended for basic style changes. **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.	string

Table 15–5 (Cont.) Exposed Applications Tree Properties

Property	Description	Allowed Values
styleClass	styleClass to use for the panelCollection component inside Applications Tree component.	string
createImmediate	Sets immediate attribute value of "Create" toolbar button and "Create" menu item.	boolean / expression
deleteImmediate	Sets immediate attribute value of "Delete" toolbar button and "Delete" menu item.	boolean / expression
duplicateImmediate	Sets immediate attribute value of "Duplicate" toolbar button and "Duplicate" menu item.	boolean / expression
editImmediate	Sets immediate attribute value of "Edit" toolbar button and "Edit" menu item.	boolean / expression
actionsMenuRendered	Sets rendered attribute value of the Actions menu. When CRUD actions are not turned on, and no af:commandMenuItem is added to the additionalActionItems facet, then actionsMenuRendered should be set to false so that an empty Actions menu would not be displayed.	boolean / expression
primaryToolbarRendered	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.	boolean / expression
secondaryToolbarRendered	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.	boolean / expression
createDisabled	Disabled attribute for create	Boolean value or EL Expression
editDisabled	Disabled attribute for edit	Boolean value or EL Expression
duplicateDisabled	Disabled attribute for duplicate	Boolean value or EL Expression
deleteDisabled	Disabled attribute for delete	Boolean value or EL Expression
confirmDelete	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.	Boolean value or EL Expression
deleteMsg	Provide a customized delete confirmation message that can be shown in the popup.	String value or EL Expression

Inline Duplicate Pattern Type

For inline patterns, the ADF tree underneath the Applications Tree should get refreshed once 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 15–7 shows a sample markup that is generated by the Applications Tree Creation wizard.

Example 15–7 Sample Markup Generated by the Applications Tree Creation Wizard

```
<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">
```

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/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/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/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 the developer needs to add custom logic to selection listener, the developer should call this default selection listener from the custom logic. treeSelectionHandler method of ApplicationsTreeBean provides the following behavior:
  - When `xxxxPatternType="secondaryWindow"` and when there is no popup configured for the level where the node needs to 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 15–8 shows sample code for calling the default selection listener from the custom selection listener.

Example 15–8 Calling the Default Selection Listener from the 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});
```

To allow developers access to some of the implementation, the Applications Tree exposes a public class, **oracle.apps.fnd.applcore.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 15–9:

Example 15–9 Defining Managed Bean Class to Attach Custom Handler to a Button

```
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;
  }
}
```

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.

15.2.1.1 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 Apps UX standards. Once you complete this wizard, you can further refine the tree definition by editing the resulting tree component as needed.

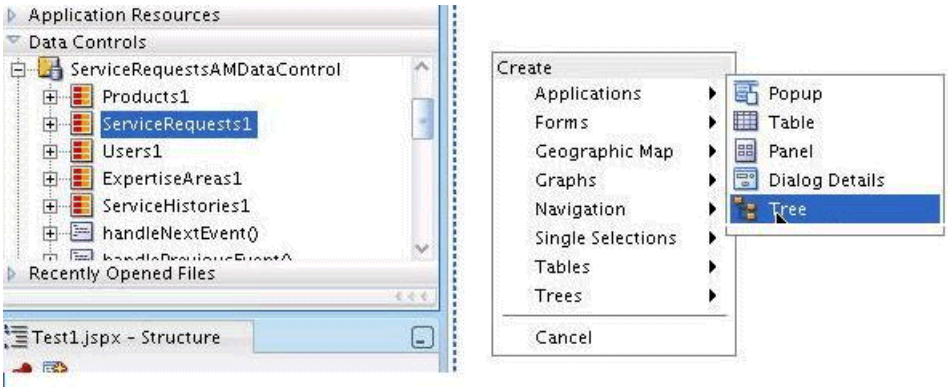
Component First

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 on to 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 15–15.

Figure 15–15 Data First Method



15.2.1.2 Applications Tree Create Wizard

The Applications Tree Create wizard consists of two panels: Create Applications Tree and Configure Tree Patterns.

Create Applications Tree Panel

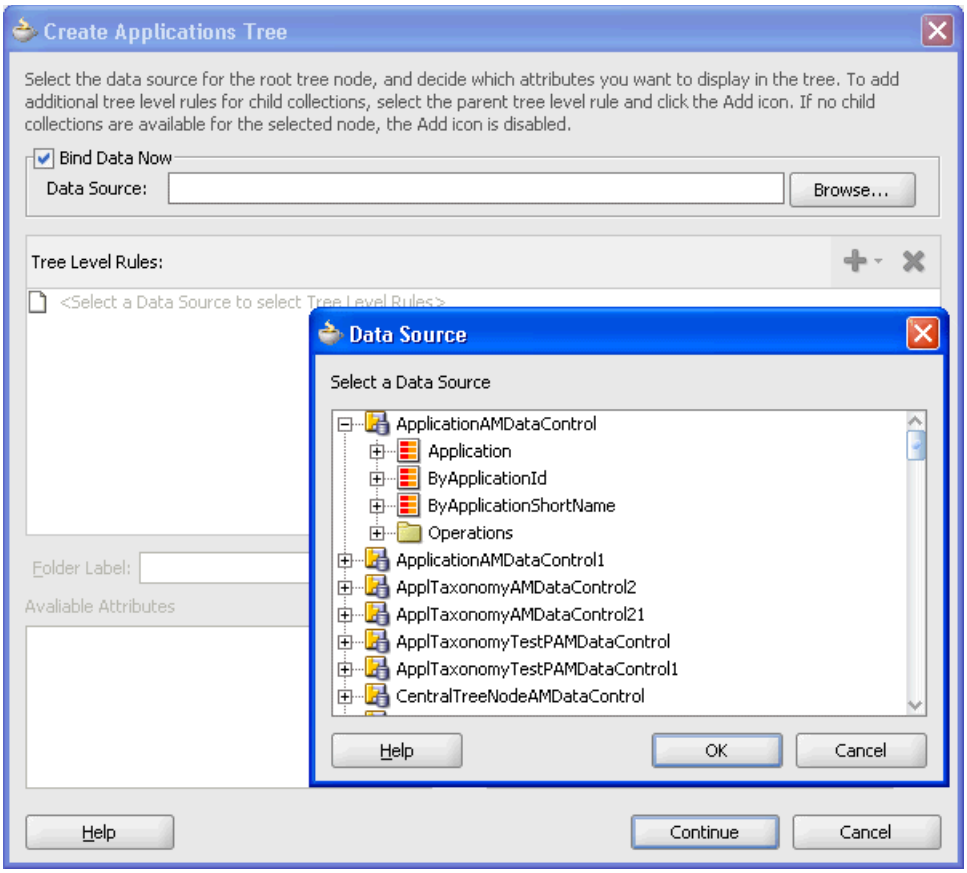
The Create Applications Tree panel will vary depending on the approach used to launch the Applications Tree creation process.

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.

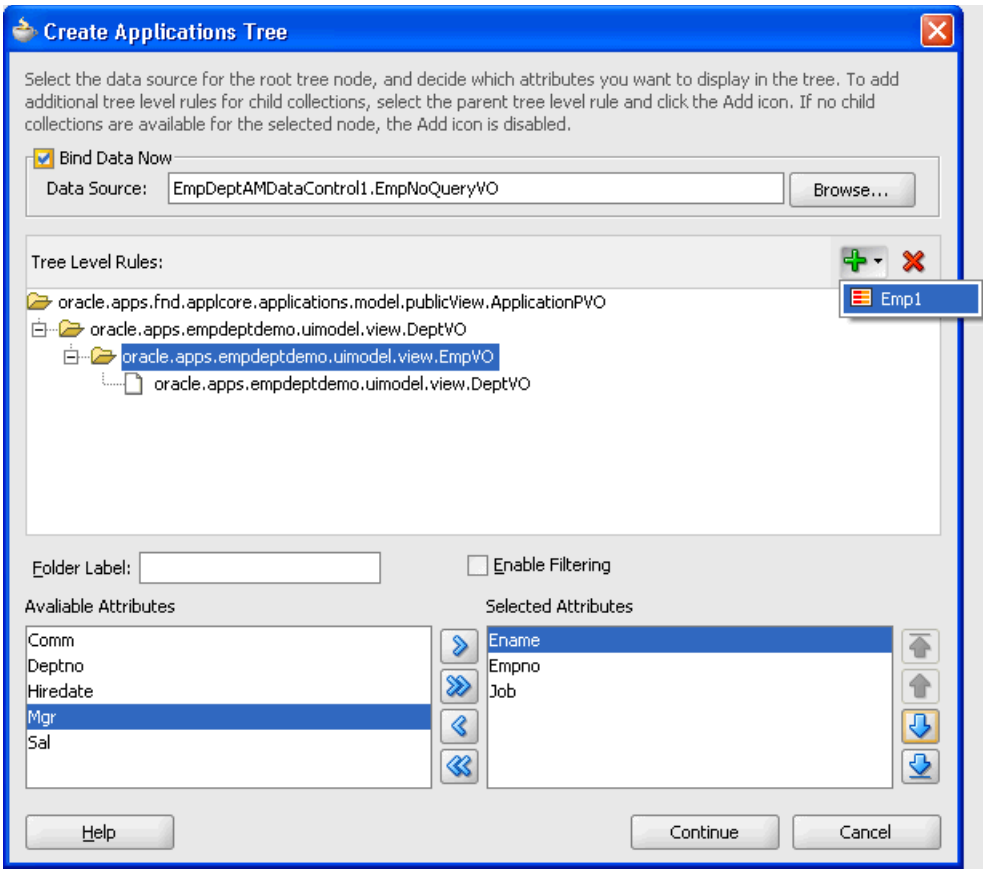
If you wish to bind a data control to the tree component using the Component First approach, check the **Bind Data Now** checkbox. This will enable the **Browse** button for the Data Source property. Click the **Browse** button to display a list of data sources available for binding. Navigate through the list, select the desired data source, and click **OK**, as shown in Figure 15–16.

Figure 15–16 Create Applications Tree Data Source



Once 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 15–17.

Figure 15–17 Configuring the ADF Tree



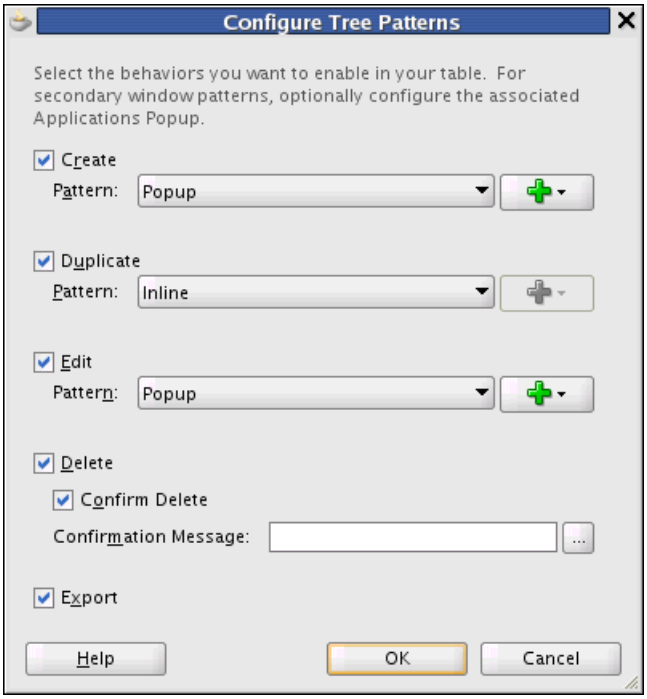
- **Tree Level Rules**: This pane displays rules that control the display order of the hierarchical tree or tree table UI components. The tree binding populates the tree UI component starting from the top of the Tree Level Rules list and continues until it reaches the last rule or until it encounters 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 EL expression that selects labels to display in the tree, such as #{label.countryLabel}.

  You also can use the EL 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 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 15–18.

Figure 15–18 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, Secondary Window, or Page).

- **Inline** - Perform the action on the current table row (only available for the Duplicate action)
- **Secondary Window** - 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 Secondary Window pattern is selected. When you click the Add button, 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 Section 15.4, "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 Section 15.1.2.2.1, "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, as shown in Figure 15–4.

When finished, click **OK** to complete creation of the Applications Tree. Selecting **Cancel** will abort the creation of the Applications Tree.

15.2.1.3 Working with the Applications Tree

This section discusses modifying settings in the JDeveloper Property Inspector. For more information, see Section 15.1.3, "Introduction to Selected Elements in the Table Property Inspector."

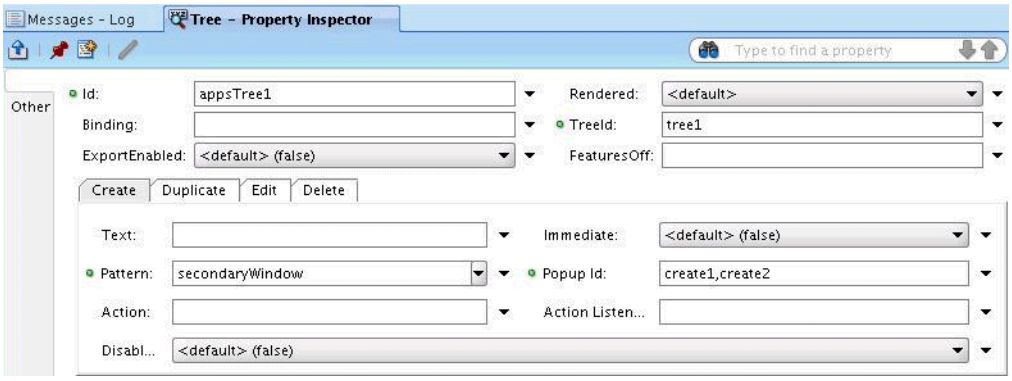
Editing - Properties

Once 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 15–19.

Figure 15–19 Tree Property Inspector



Adding a Data Source

Once 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 "Organizing Content on Web Pages" and "Customizing the Appearance Using Styles and Skins" in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

15.3 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 15–6 are exposed on the Applications Tree Table:

Table 15–6 Applications Tree Table Properties

Property	Description	Allowed Values
`id`	Unique ID for this Applications Tree Table.	string
`rendered`	Whether the Applications Tree Table is rendered or not.	boolean / expression
`treeTableId`	Unique ID of the ADF tree table underneath this Applications Tree Table.	string
`createPatternType`	Whether any Create pattern is enabled, and if yes, which pattern.	none, inline, secondaryWindow (then Create Popup Id must also be set), page
`editPatternType`	Whether any Edit pattern is enabled, and if yes, which pattern.	none, inline, secondaryWindow (then Edit Popup Id must also be set), page
`duplicatePatternType`	Whether any Duplicate pattern is enabled, and if yes, which pattern.	none, inline (see Note), secondaryWindow (then Duplicate Popup Id must also be set), page
`deleteMsg`	Provide a customized delete confirmation message that can be shown in the popup.	Boolean value or EL Expression
`deleteEnabled`	Whether any Delete pattern is enabled.	boolean
`createEnabled`	Whether any Create pattern is enabled.	boolean
`duplicateEnabled`	Whether any Duplicate pattern is enabled.	boolean
`editEnabled`	Whether any Edit pattern is enabled.	boolean
`confirmDelete`	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.	Boolean value or EL Expression
`createAction`	Action binding for the Create icon.	method expression
`editAction`	Action binding for the Edit icon.	method expression
`duplicateAction`	Action binding for the Duplicate icon.	method expression
`deleteAction`	Action binding for the Delete icon.	method expression
`createActionListener`	Action listener binding for the Create icon.	method expression
`editActionListener`	Action listener binding for the Edit icon.	method expression
`duplicateActionListener`	Action listener binding for the Duplicate icon.	method expression
`deleteActionListener`	Action listener binding for the Delete icon.	method expression
`createPopupId`	ID of the popup to be invoked when the Create button is clicked.	string
`editPopupId`	ID of the popup to be invoked when the Edit button is clicked.	string
`duplicatePopupId`	ID of the popup to be invoked when the Duplicate button is clicked.	string
`deletePopupId`	ID of the popup to be invoked when the Delete button is clicked.	string
`createText`	Overrides the default label for the Create menu item.	expression

Table 15–6 (Cont.) Applications Tree Table Properties

Property	Description	Allowed Values
`editText`	Overrides the default label for Edit menu item.	expression
`duplicateText`	Overrides the default label for Duplicate menu item.	expression
`deleteText`	Overrides the default label for Delete menu item.	expression
`exportEnabled`	Whether export is enabled.	boolean / expression
`featuresOff`	A list of default features to turn off for the panelCollection, for instance detach (see featuresOff attribute of panelCollection for more details)	string
`inlineStyle`	The CSS styles to use for the panelCollection component inside the Applications Tree Table component. This is intended for basic style changes. **Note:** *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.	string
`styleClass`	styleClass to use for the panelCollection component inside the Applications Tree Table component.	string
`createImmediate`	Sets immediate attribute value of Create toolbar button and Create menu item.	boolean / expression
`deleteImmediate`	Sets immediate attribute value of Delete toolbar button and Delete menu item.	boolean / expression
`duplicateImmediate`	Sets immediate attribute value of Duplicate toolbar button and Duplicate menu item.	boolean / expression
`editImmediate`	Sets immediate attribute value of Delete toolbar button and Delete menu item.	boolean / expression
createDisabled	Disabled attribute for Create.	Boolean value or EL Expression
deleteDisabled	Disabled attribute for Delete.	Boolean value or EL Expression
editDisabled	Disabled attribute for Edit.	Boolean value or EL Expression
duplicateDisabled	Disabled attribute for Duplicate.	Boolean value or EL Expression
`actionsMenuRendered`	Sets the rendered attribute value of the Actions menu. When CRUD actions are not turned on, and no af:commandMenuItem is added to the additionalActionItems facet, actionsMenuRendered should be set to False so that an empty Actions menu will not be displayed.	boolean / expression
`primaryToolbarRendered`	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.	boolean / expression

Table 15–6 (Cont.) Applications Tree Table Properties

Property	Description	Allowed Values
`secondaryToolbarRendered`	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.	boolean / expression
`toggleEditRendered`	The toggleEditRendered feature is used to render the editAll or clickToEdit choices for Applications Tree Table. See Section 15.3.1.2.4, "Toggle Click to Edit / Edit All in Applications Tree Table."	Boolean. Default value is **False**.

Table 15–6 (Cont.) Applications Tree Table Properties

Property	Description	Allowed Values
`<button_ name>PartialTriggers` For example: `deletePartialTriggers`	Partial triggers attribute for the <button_ name> toolbar button. 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 commandToolbarButton 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.	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 editDiabled, deleteDisabled or duplicateDisabled property of the Applications Tree Table. `#{bindings.VOiterator.estimated RowCount == 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 once 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 or not 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.input Value == '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.

Note: For inline patterns, the ADF tree table beneath the Applications Tree Table should be refreshed once 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 15–10 shows a sample markup that is generated by the Applications Tree Table Creation wizard.

Example 15–10 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">
```

Table 15–7 shows the facets that are exposed on the Applications Tree Table.

Table 15–7 Applications Tree Table Facets

Facet	Description	Allowed Children
`treeTable`	Facet for holding the ADF tree table.	ADF Tree Table
`additionalToolbarButtons`	Facet for adding toolbar icons by the developer.	ADF Command Toolbar Buttons
`additionalActionItems`	Facet for adding more menu items to default menu items.	ADF menu item component
`appsTreeTableAfterToolbar`	Facet for adding an af:toolbar or af:groups of af:toolbars that appear in a separate row below the normal group of toolbars.	ADF Toolbar or ADF Groups of Toolbar
`appsTreeTableSecondaryToolbar`	Facet for adding more commandToolbar components to the secondary toolbar.	ADF Command Toolbar Button component

Table 15–7 (Cont.) Applications Tree Table Facets

Facet	Description	Allowed Children
`appsTreeTableStatusbar`	Facet for adding component containing statusbar items. These statusbar items are merged with standard items provided by the panelCollection.	ADF component
`appsTreeTableViewMenu`	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.	ADF menu item component
`popup`	Facet for adding popups. See Section 15.4, "Using the Custom Wizard with Applications Popups." **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.	Any number of popups under a layout component.

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 needs to 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 15–11.

Example 15–11 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});
```

To allow Applications developers access to some of the implementation, the Applications Tree Table exposes a public class `oracle.apps.fnd.applcore.patterns.ApplicationsTreeEventHandler` 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.

Example

To attach a custom button handler to the **Create** button:

1. Define a managed bean class, as shown in Example 15–12.

Example 15–12 Define a Managed Bean Class to Attach Custom Handler to a Button

```
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;
  }
}
```

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.

15.3.1 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. Once you complete this wizard, you can further refine the tree definition by editing the resulting tree component as needed.

Valid drop locations in the page or page fragment include: ADF Form, 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, and af:pageTemplate#<localArea_Facet>).

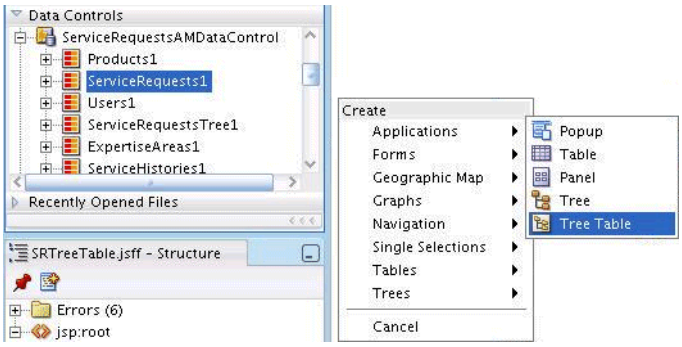
Component First

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 15–20.

Figure 15–20 Data First Method to Add a Tree Table



15.3.1.1 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

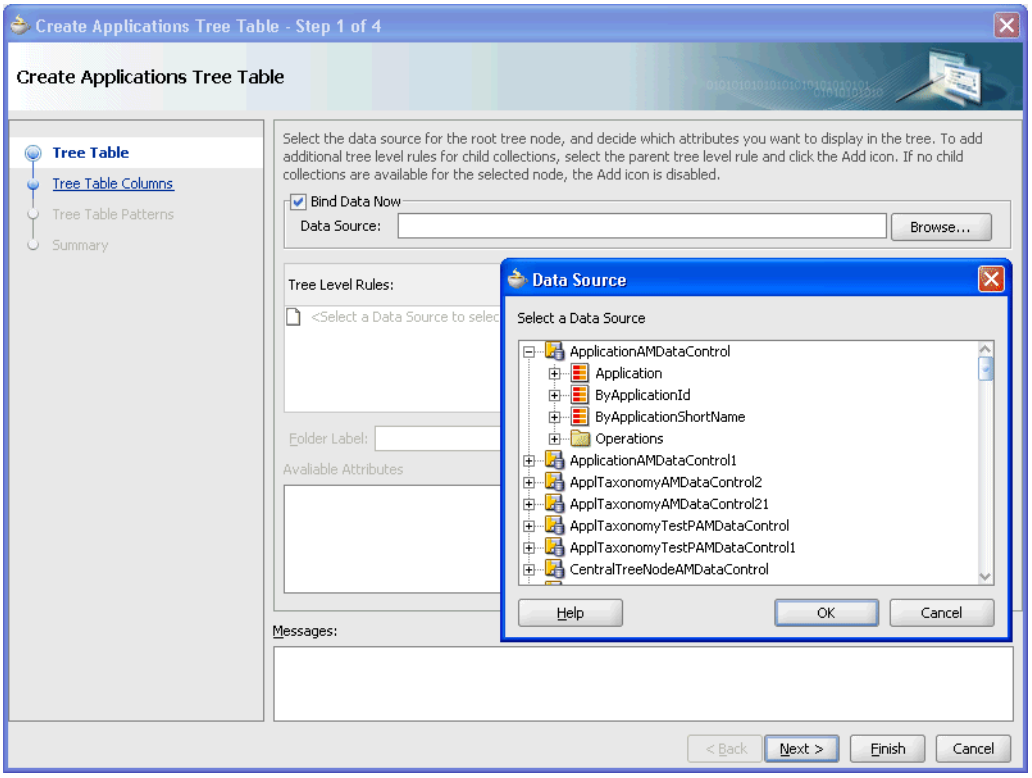
This step creates a tree binding and node definitions of the tree. The Create Applications Tree Table Panel will vary depending on the approach used to launch the Applications Tree Table creation process.

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.

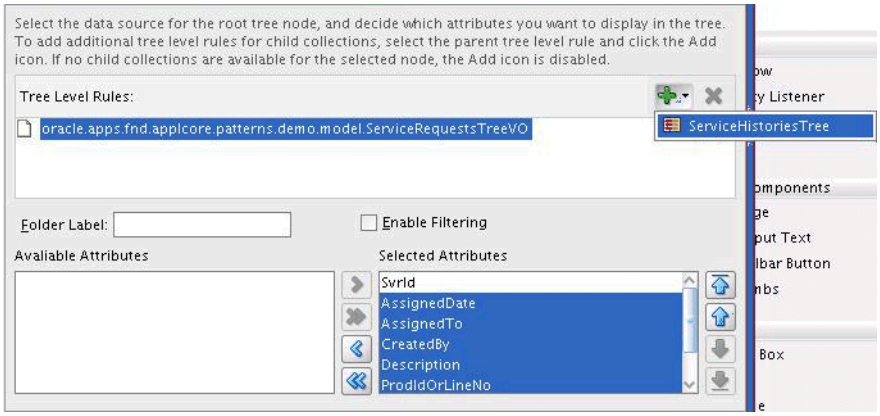
If you wish to bind a data control to the tree component using the Component First approach, select the Bind Data Now checkbox. This will enable the Browse button for the Data Source property. Click the Browse button to display a list of data sources available for binding. Navigate through the list and select the desired data source. Click the OK button, as shown in Figure 15–21.

Figure 15–21 Create Applications Tree Table



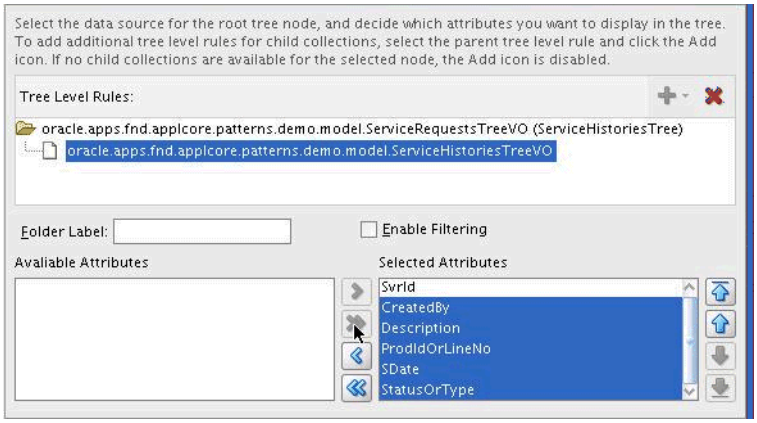
Once the Data Source is selected, you can configure the ADF tree. Click the **Add** icon to add one of the children of the selected Data Source to be the next level of the tree, as shown in Figure 15–22.

Figure 15–22 Adding a Child of the Data Source



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 tree node or columns in the tree table, as shown in Figure 15–23.

Figure 15–23 Selecting Attributes

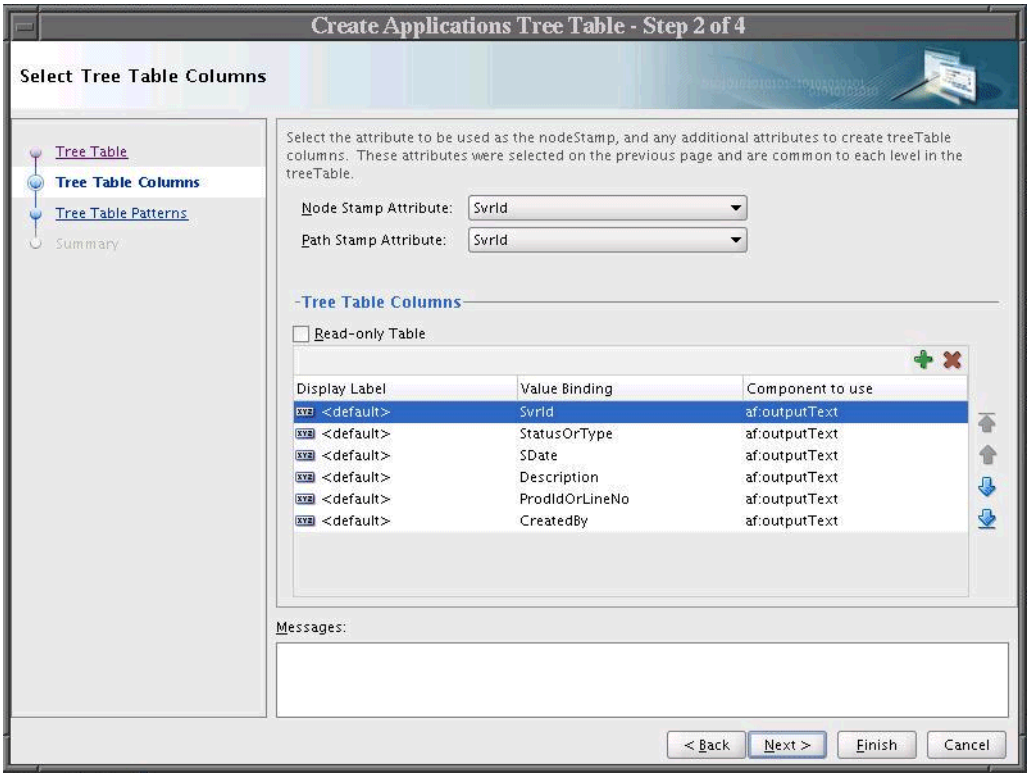


When finished, click **Next** to proceed to the Select Tree Table Columns panel. Select **Cancel** to abort the creation of the Applications Tree Table.

Select Tree Table Columns Panel

The Select Tree Table Columns panel shown in Figure 15–24 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.

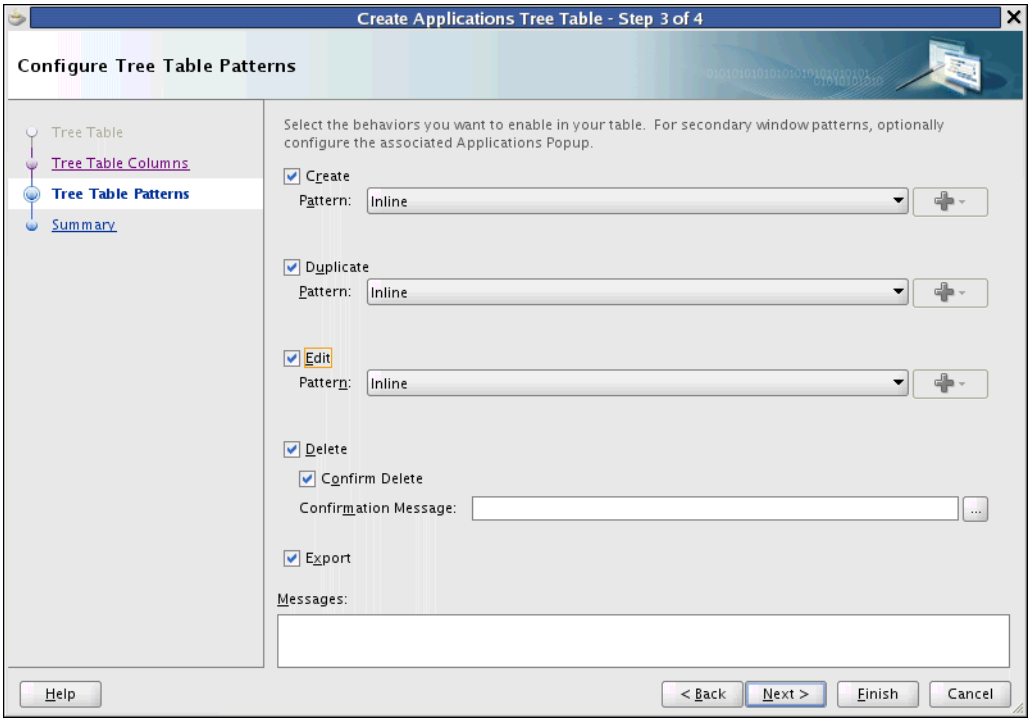
Figure 15–24 Select Tree Table Columns



Configure Tree Table Patterns Panel

The Configure Tree Table Patterns panel allows you to select the default actions offered by your Applications Tree Table, shown in Figure 15–25.

Figure 15–25 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, Secondary Window, Page).

- **Inline** - Perform the action on the current table row.
- **Secondary Window** - 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 Secondary Window 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 Section 15.4, "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 Section 15.4, "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 Section 15.1.2.2.1, "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, as shown in Figure 15–4.

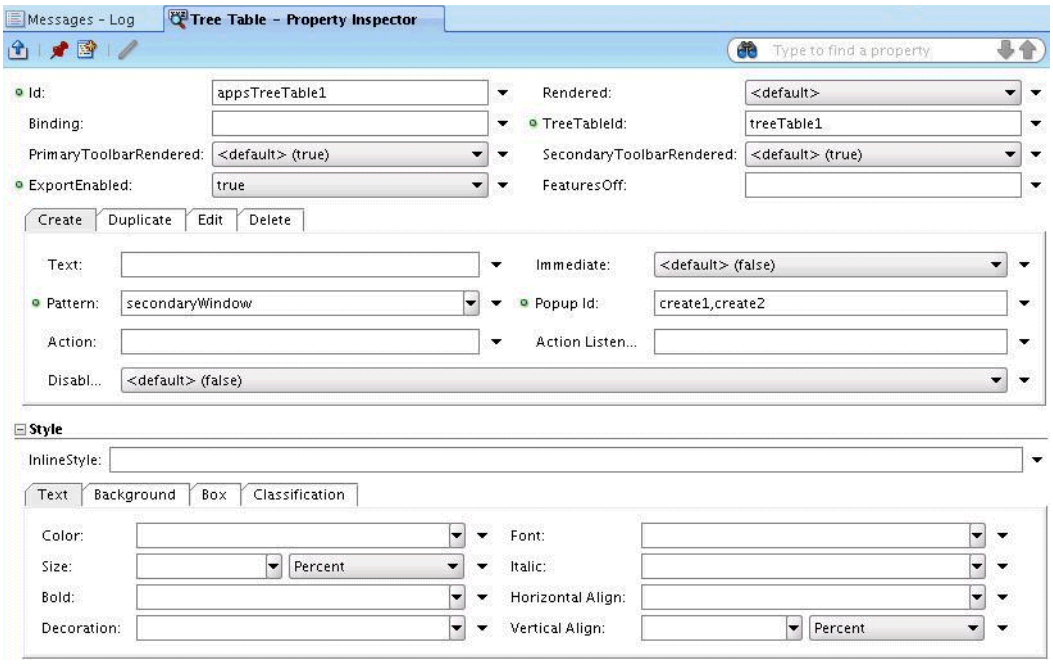
Click **Finish** to complete creation of the Applications Tree Table. Select **Cancel** to abort the creation of the Applications Tree Table.

15.3.1.2 Working with the Applications Tree Table

Once 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 15–26.

Figure 15–26 Tree Table Property Inspector



15.3.1.2.1 Adding a Data Source

Once you have created the Applications Tree Table, you can add data controls to the facets / 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.

15.3.1.2.2 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.

15.3.1.2.3 Increasing Tree Table Width to Fill 100% of Its Container 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 "Organizing Content on Web Pages" and "Customizing the Appearance Using Styles and Skins" in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

15.3.1.2.4 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 Section 15.1.2.2.3, "Toggle Click to Edit / Edit All in Applications Table."

15.4 Using the Custom Wizard with Applications Popups

> **Note:** `af:popup` is a generic function documented in the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*. 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 15–8.

Table 15–8 Patterns That Require Popups

Pattern Set	Patterns	Description/User Action
Attachments	Attachments Field, Attachments Column	When users display attachments, the resulting popup displays the current attachment, and also allows users to add new attachments to an entity.
Compare Objects	Configure Comparison, One to Many, Configure and Compare	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.
Create	Create Multiple Objects	When users click Create Multiple Objects or make a choice from a menu or toolbar, the resulting popup allows users to create multiple objects.
Detail On Demand	Popup Details	When users select a table row and click an information button, the resulting popup displays information about the selected row.
Edit	Secondary Window Edit	When users select records and click Edit on a table toolbar, the resulting popup allows users to edit those records.

Table 15–8 (Cont.) Patterns That Require Popups

Pattern Set	Patterns	Description/User Action
Information Entry Form	Secondary Window	When users click a button, the resulting popup allows users to enter data into the popup.
Record Navigation	Secondary Window Detail	When users navigate through data records, the resulting popup allows users to edit the records.
Transactional Search/Results	Popup Window	When users click Search within an application and enter search criteria, the popup displays the search results.

15.4.1 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.

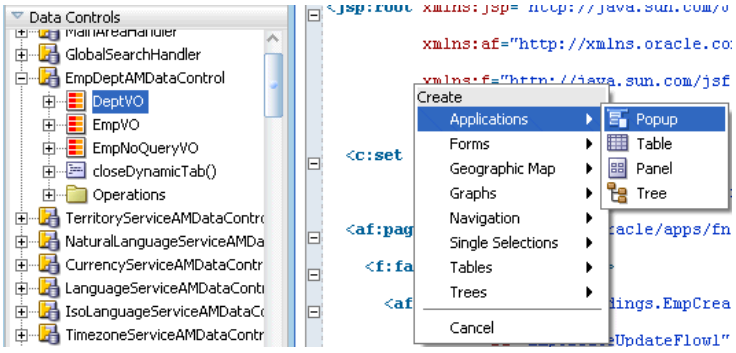
15.4.1.1 How to Add Applications Popups to JSF Pages or Page Fragments

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 15–27.

Figure 15–27 Dragging a Data Control to the JSF Page



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 Section 15.4.1.2, "How to Add Applications Popup Components Using the Wizard."

> **Note:** The data source for the table becomes the default data source for the popup.

15.4.1.2 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:

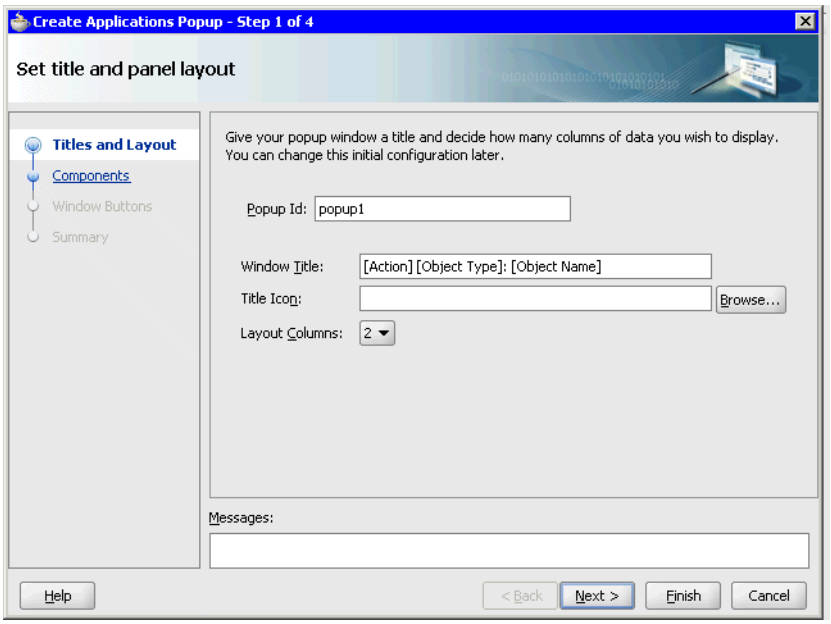
- Displays a preview of the popup.
- Creates the popup with the values you provided on that screen and any previous screens, and default values for the remaining screens.

> **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 15–28.

Figure 15–28 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:

     ```
     [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 "Internationalizing and Localizing Pages" in the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*, 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: `#{af:formatNamed(bundle.EDIT_INVOICE,'INVOICE_NUMBER',bindings.Invoices.InvoiceNumber)}`

     but in the resource bundle, it would be defined as:

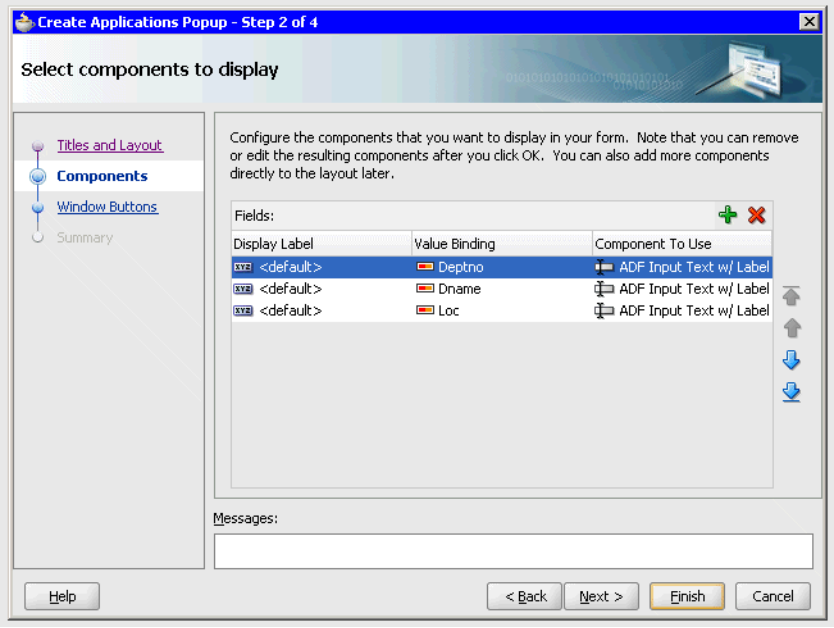
     ```
     <trans-unit id="EDIT_INVOICE">
         <source>Edit Invoice: {INVOICE_NUMBER}</source>
         <target/>
     </trans-unit>
     ```

   - [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.

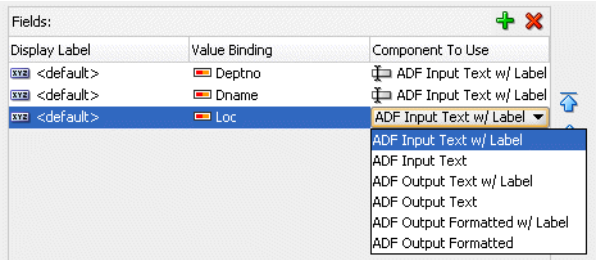
2. Click **Next** to display the **Select Components to Display** dialog, as shown in Figure 15–29.

Figure 15–29 Select Components to Display Dialog



- 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 15–30.

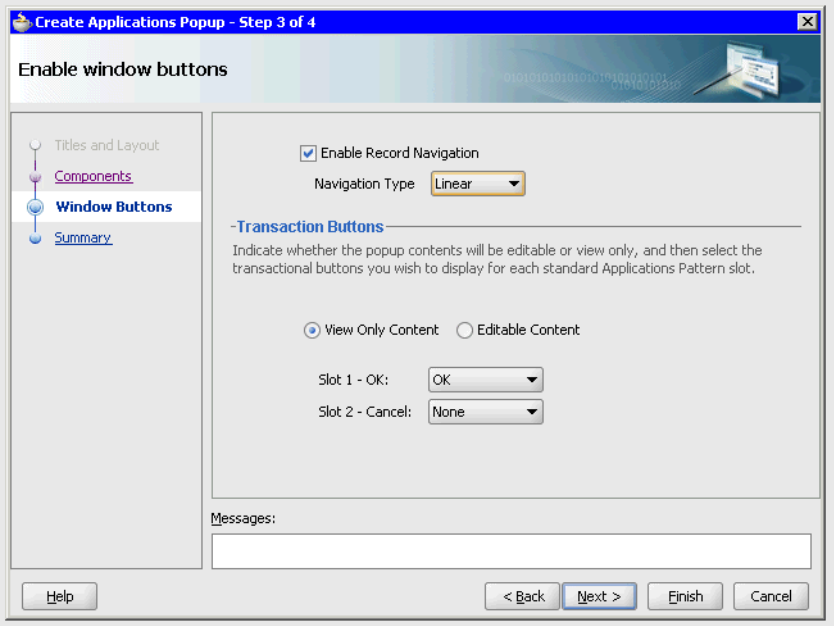
Figure 15–30 Components to Use List



See the description of "Components to Use" in Using Attributes to Create Text Fields in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

- Click **Next** to display the Enable Window Buttons dialog, as shown in Figure 15–31.

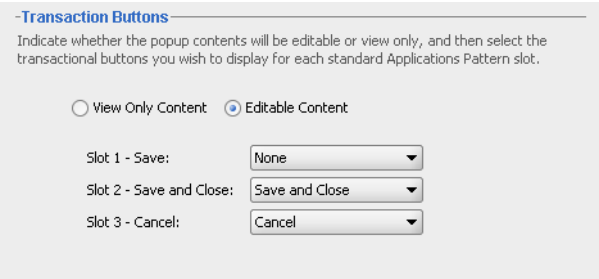
Figure 15–31 Enable Window Buttons Dialog



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.

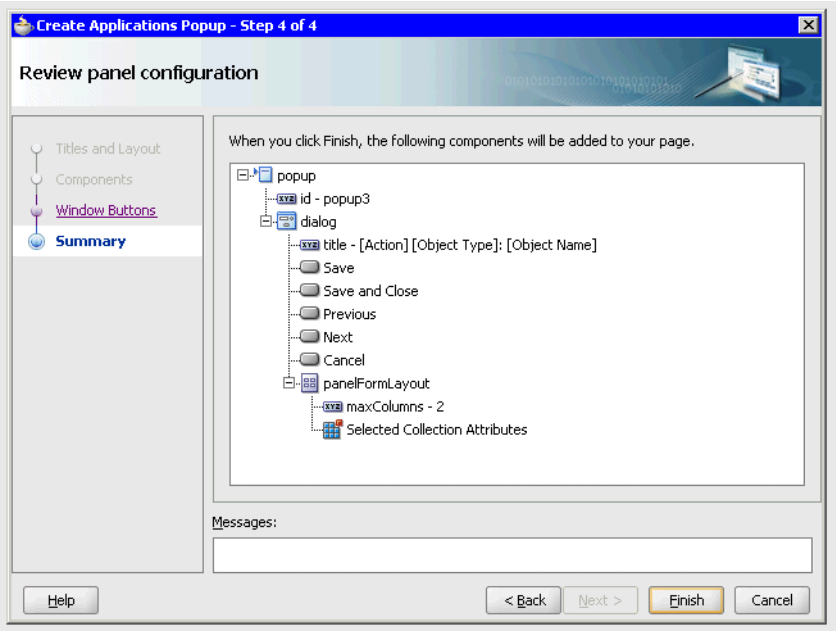
     When you choose View Only Content, you automatically enable Slots 1 and 2. When you choose Editable Content, you automatically enable Slots 1, 2, and 3, as shown in Figure 15–32.

Figure 15–32 Editable Content Slots



- Click **Next** to display the Review Panel Configuration dialog, as shown in Figure 15–33.

Figure 15–33 Review Panel Configuration Dialog



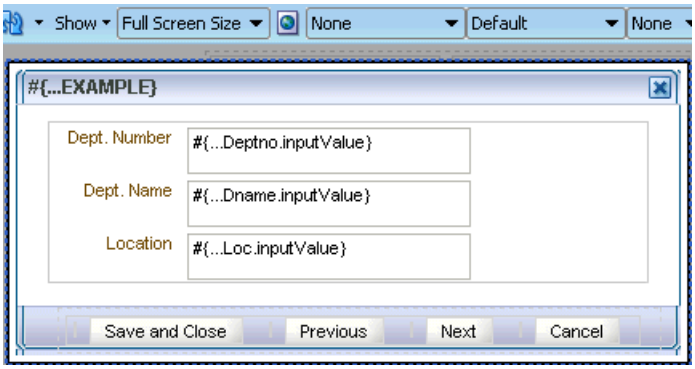
4. The dialog displays a list of your choices.

   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 15–34.

Figure 15–34 Page Displaying Popup and Popup Components



15.4.2 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.

15.4.2.1 Accessing 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.

15.4.2.2 Adding 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.

15.4.2.3 Adding 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.

15.4.2.4 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.

16

Implementing Applications Panels, Master-Detail, Hover, and Dialog Details

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:

- Section 16.1, "Implementing Applications Panels"
- Section 16.2, "Implementing Applications Master-Detail"
- Section 16.3, "Implementing Hover"
- Section 16.4, "Implementing Applications Dialog Details"

For basic information, see:

- Chapter 13, "Getting Started with Your Web Interface"
- Chapter 14, "Implementing the UI Shell"
- Chapter 15, "Implementing UIs in JDeveloper with Application Tables, Trees and Tree Tables"

16.1 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.

16.1.1 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 16–7.

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 16–1 contains attributes that are exposed for the buttons.

Table 16–1 Attributes of Standard Panel Buttons

Property	Description	Data Type
`id`	Unique identification number for the panel.	`string`
`rendered`	Whether the panel is rendered (that is, converted from an object-based description into a graphical image for display).	`boolean or expression`
`title`	Panel title.	`string or expression`
`navigationType`	Type of navigation for that panel.	Navigation types: ▪ `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. ▪ `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. ▪ `none` - navigation is disabled.
`<button_name>Visible` For example: `submitVisible`	Whether the button is visible in the UI.	`boolean or expression`
`<button_name>Rendered` For example: `submitRendered`	Whether the button is rendered in the UI (that is, converted from an object-based description into a graphical image for display).	`boolean or expression`
`<button_name>Action`	Type of action that the button performs.	`EL expression`
`<button_name>PopupId` For example: `submitPopupId`	ID of the popup that appears when users press the button.	`string`
`<button_name>ShortDesc` For example: `previousShortDesc`	Tooltip text for the button.	`string`

Table 16–1 (Cont.) Attributes of Standard Panel Buttons

Property	Description	Data Type
`<button_name>Disabled` For example: `submitDisabled`	**Next** and **Previous** buttons only: Whether the button should be disabled in the UI.	`boolean or expression`
`submitText`	Submit button text: Text associated with the **OK** button.	`string or expression`
`scrollable`	Sets to true when scroll bar needs to be enabled. When scrollable is set to true, it sets `layout="scroll"` on the `af:panelGroupLayout` component inside Applications Panel. Thus, the developer does not need to place `af:panelGroupLayout` with `layout="scroll"` directly under the Applications panel.	`boolean / expression`
`saveOptionsStyle`	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.	`button` or `dropButton`

`instructionText`	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.	The `instructonText` attribute can take a String value or an ELExpression. A helpTopicID can be passed to this attribute as an ELExpression. For example:
		`instructionText="#{adfFacesContext.helpProvider'helpTopicId'].instructions}"`
		or
		`instructionText="#{adfFacesContext.helpProvider['helpTopicId'].definition}"`
		or
		`instructionText="#{adfFacesContext.helpProvider['helpTopicId'].externalUrl}"`

`panelToolbarRendered`	If no default buttons that are provided by the Applications Panel are used, set this value to false to avoid displaying unnecessary separators.	Boolean
`revertImmediate`	Sets the **immediate** attribute on the Revert button. Sets 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 the Apply Request Values phase of the request processing lifecycle, rather than waiting until the Invoke Application phase.	Boolean. Default is **false**.

Table 16–1 (Cont.) Attributes of Standard Panel Buttons

Property	Description	Data Type
`submitStyle`	Sets the appearance of the Submit button. The Submit button can be rendered as a normal button, or as a drop button, depending on the value of this attribute. When it's set to `dropButton`, the developer is expected to have other submit options turned on, or add `af:commandMenuItem` to the `submitButtonMenu` facet.	String. The two values are button (the default) and dropButton.
`previousPartialSubmit`	Sets the `partialSubmit` attribute on the **Previous** button.	True or false (default)
`nextPartialSubmit`	Sets the `partialSubmit` attribute on the **Next** button.	True or false (default)
`saveAndCreateAnotherText`	Sets the text that is displayed on the **saveAndCreateAnother** button.	String
`createAnotherText`	Sets the text that displays on the **createAnother** button.	String
`<button_name>Action`	Sets the action attribute on the button with **<button_name>**. Users must provide their own action; there is no default action.	String or EL Expression.
`<button_name>ActionListener`	Sets the actionListener attribute on the button with **<button_name>**. Users must provide their own `actionListener`; there is no default `actionListener`.	EL Expression.
`<button_name>PartialTriggers` For example: `saveAndClosePartialTriggers`	`partialTriggers` attribute for **<button_name>** button.	String or EL Expression. **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].

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 16–2 contains facets that are exposed for each panel button.

Table 16–2 Facets of Standard Panel Buttons

Facet	Description	Allowed Children
`contents`	Facet for holding developer-defined content or content generated at design time.	Any ADF component.
`navigationBar`	Facet for holding the navigation choice list if the chosen Record Navigation Type is non-linear.	`ADF selectOneChoice`
`actionButtonBar`	Facet for holding custom action buttons.	`ADF commandButtons` and `commandToolbarButtons` under some ADF layout components.

Table 16–2 (Cont.) Facets of Standard Panel Buttons

Facet	Description	Allowed Children
`saveButtonMenu`	Facet for holding the custom menu and menu items for the **Save** button.	`af:commandMenuItem` or `af:group`
`submitButtonMenu`	Facet for holding the custom menu and menu items of the **Submit** button.	None
`popup`	Facet for holding any popups required for any of the buttons.	Applications popups under some ADF layout components.
`appsPanelLegend`	Facet for displaying legend information on the header.	
`appsPanelContext`	Facet for displaying context information next to the header. The contextual information is displayed next to the header text.	
`customSaveDropButton`	Facet for adding custom Save drop button.	This facet should contain `<af:group>` with `<af:commandToolbarButton>` under it. The Design Time handles this for you. See Figure 16–8 and its description.
`localContext`	Facet for adding content into local context region.	

Table 16–2 (Cont.) Facets of Standard Panel Buttons

Facet	Description	Allowed Children
`taskStamp`	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: `<af:panelGroupLayout layout="vertical" valign="top" styleClass="AFStampContainer" id="ptpgl5"> <af:panelGroupLayout layout="horizontal" halign="end" id="ptpgl6"> <af:outputText value="Last Updated" styleClass="TaskStampTextLabel" id="ptot8"/> <af:outputText value="08-Nov-2007" styleClass="AFTaskStampTextValue" id="ptot9"/> </af:panelGroupLayout> <af:panelGroupLayout layout="horizontal" halign="right" id="ptpgl7"> <af:outputText value="Budget Remaining" styleClass="TaskStampTextLabel" id="ptot10"/> <af:outputText value="$20,000.00" styleClass="AFTaskStampTextValue" id="ptot11"/> </af:panelGroupLayout> </af:panelGroupLayout>`	

Table 16–2 (Cont.) Facets of Standard Panel Buttons

Facet	Description	Allowed Children

`collabrationToolbar`	Facet for adding collaboration toolbar buttons.	**Example**:
		`<f:facet name="collaborationToolbar">`
		`<af:toolbox>`
		`<af:toolbar>`
		`< af:commandImageLink text="One" icon="/image1" id="mycmd1"/>`
		`< af:commandImageLink text="Two" icon="/image2" id="mycmd2"/>`
		`< af:commandImageLink text="Three" icon="/image3" id="mycmd3"/>`
		`</af:toolbar>`
		`</af:toolbox>`
		`</f:facet>`

`scalingInfo`	Facet for adding scaling information.	Example:
		`<af:panelGroupLayout layout="vertical"`
		`styleClass="AFStampContainer" id="pgl3">`
		`<af:outputText value="AUD = Australian Dollar" id="ot5"/>`
		`</af:panelGroupLayout>`
		Example for scalingInfo with more than one value:
		`<af:panelGroupLayout layout="vertical"`
		`styleClass="AFStampContainer" id="pgl3">`
		`<af:outputText value="AUD = Australian Dollar | Amounts in thousands" id="ot5"/>`
		`</af:panelGroupLayout>`

`submitButtonMenu`	Facet for holding the custom menu and menu items for the **Submit** button.	af:`commandMenuItem` or `af:group`

Table 16–2 (Cont.) Facets of Standard Panel Buttons

Facet	Description	Allowed Children
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: `<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" createText="#{viewcontrollerBundle.NEW}">`	Scroll (the default) or Stretch.
appsPanelTrain	Facet for adding a horizontal train above header.	

Table 16–2 (Cont.) Facets of Standard Panel Buttons

Facet	Description	Allowed Children
`contentsFacet`		This can be either `scroll` (the default) or `stretch`. Set to `stretch` when using the `contentsStretch` facet.

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.applcore.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.

Custom Button Handling

Follow these steps to attach a custom button handler to the **Cancel** button.

1. Define the managed bean class, as shown in Example 16–1.

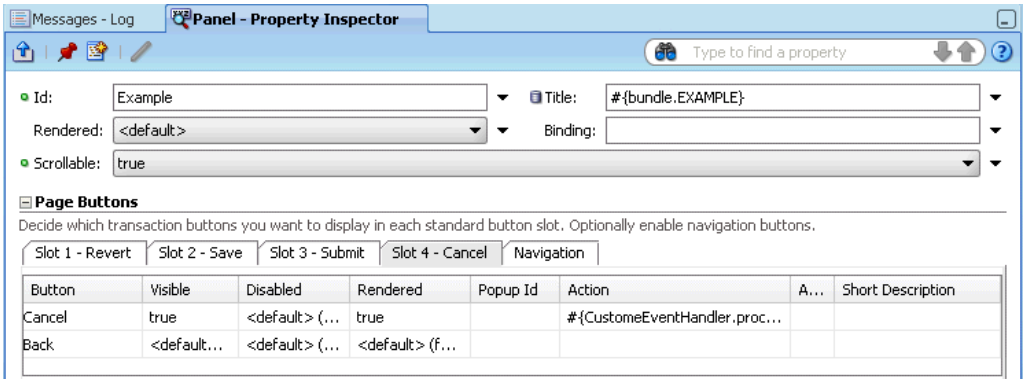
Example 16–1 Example of Attaching a Custom Handler to a Button

```
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;
 }
```

```
}
```

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 16–1, set `#{CustomeEventHandler.processCancel}` as the expression for the property.

Figure 16–1 Table Property Inspector



4. Click the **Create** tab.
5. In the **Create Action** expression field, enter the following expression:

   ```
   #{CustomEventHandler.processCreate}
   ```

6. Click **Set**.

16.1.2 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.

16.1.2.1 Adding 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:

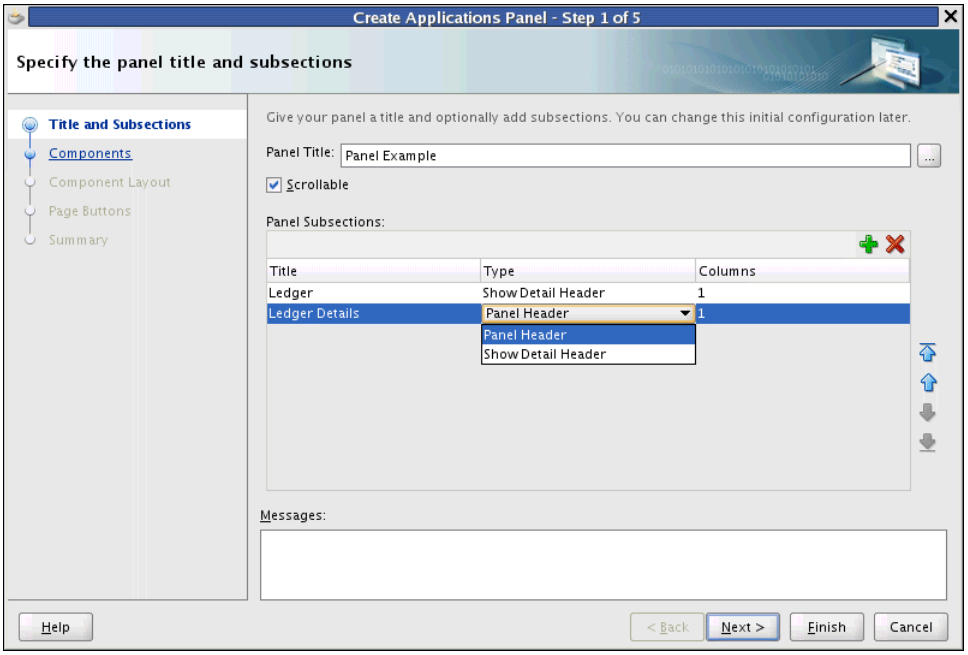
- Specify panel titles and subsections
- Select panel components
- Bind a data source to the panel
- Arrange panel components
- Select panel buttons

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 16–2.

Figure 16–2 Specifying the Panel Title and Subsections



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 "Internationalizing and Localizing Pages" in the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

- 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.

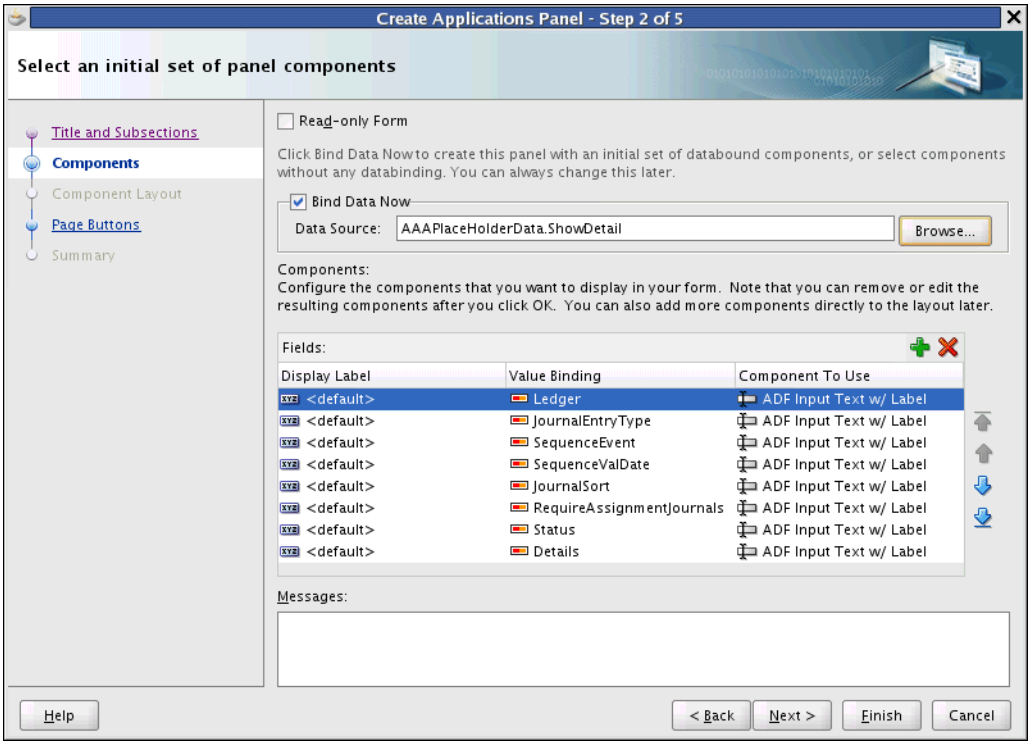
  The Panel Subsections is used to divide the Applications Panel facet (contents) with other layout components, such as `panelHeader`, show detail header, and panelGroupLayout. This lets the developer decide the layout during Design Time without needing to add each of these layouts manually after the panel 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 16–3.

Figure 16–3 Selecting an Initial Set of Panel Components



3. In the Panel Components dialog:

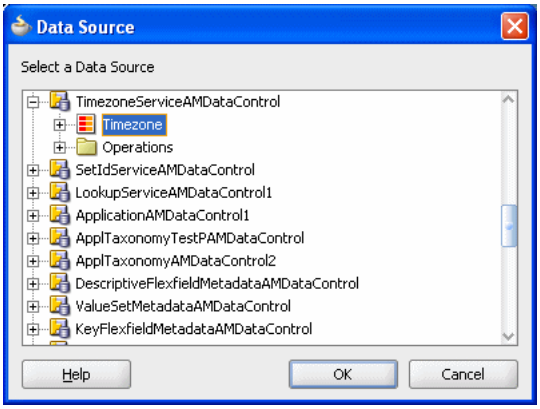
a. Click **Read-only Form** to create a read-only form. (optional) If you select Read-only, the choices in the Component to Use column will change from Input Text to Output Text types.

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 16–4.

Figure 16–4 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 Section 16.1.3.4, "Adding a Data Source to an Existing Panel."

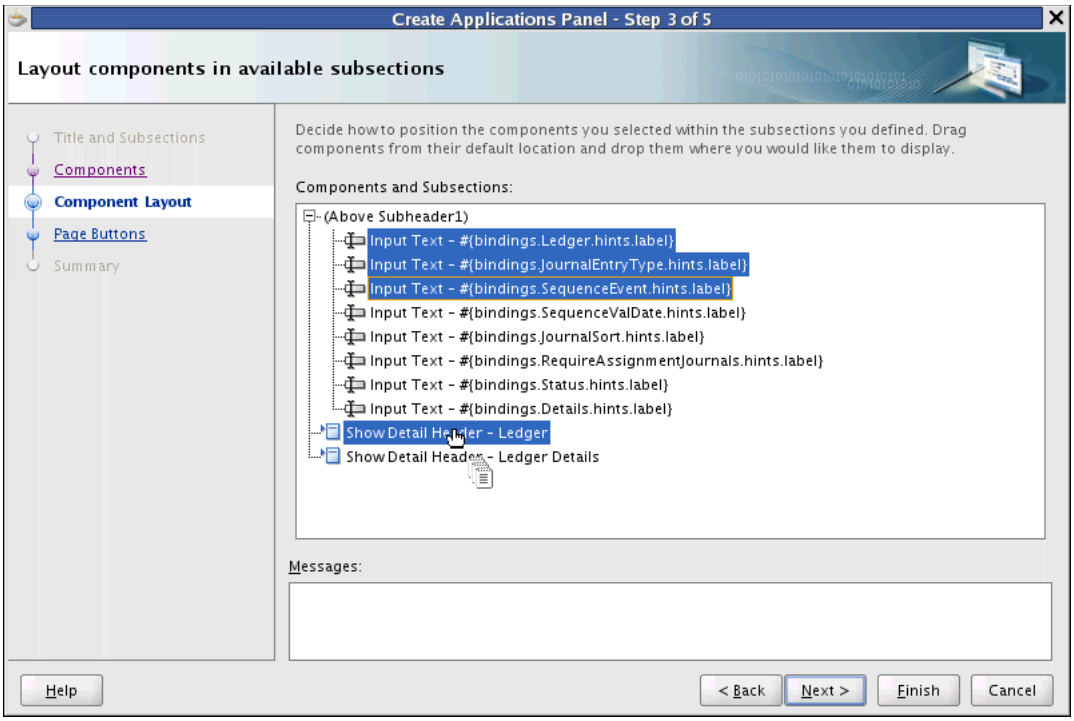
d. **Display Label:** In general, the labels defined in the selected Data Control will be what you want and you can leave this value at the default <Default> setting. Otherwise, enter a new label name.

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 or not you choose read-only. If you selected Read-only, the choices will change from Input Text to Output Text types.

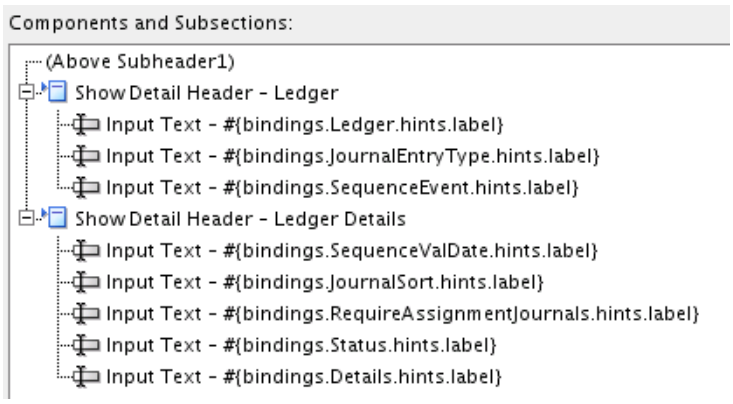
4. Click **Next**. The Components Layout dialog is displayed, as shown in Figure 16–5.

Figure 16–5 Layout Components In Available Subsections Dialog



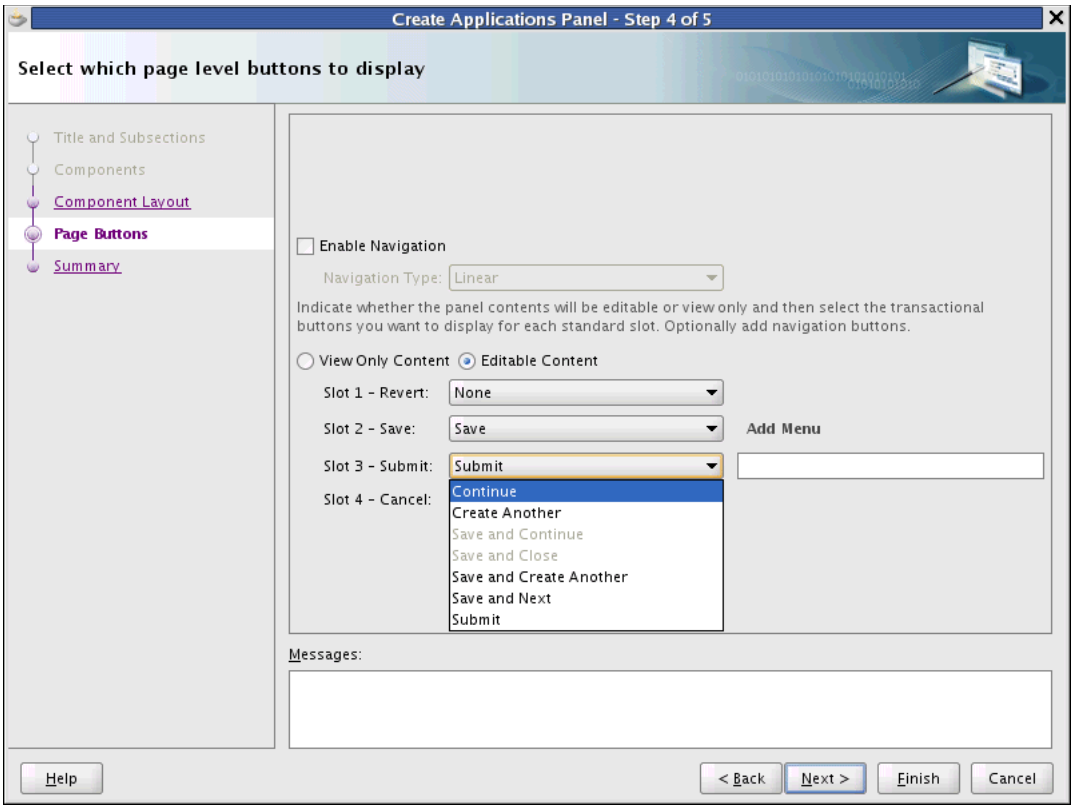
5. Drag components from their default locations to your previously defined subsections. The result will appear similar to Figure 16–6.

Figure 16–6 Example of Component Layout



6. Click **Next**. The Page Buttons dialog is displayed, as shown in Figure 16–7.

Figure 16–7 Select Page-Level Buttons Dialog



7. In the Page Level Buttons dialog:
   - To enable panel navigation (optional):
     - Select **Enable Navigation**.
     - Choose a navigation type (Linear or Non-Linear).

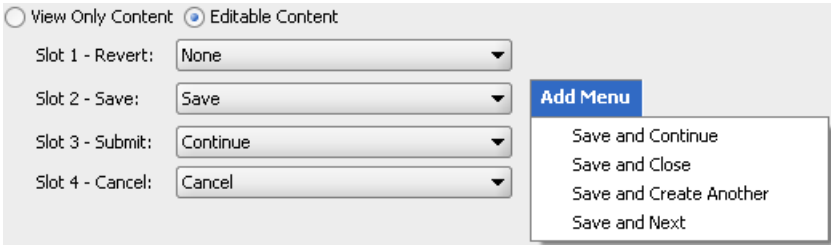
       Linear sends users to an adjacent or contiguous window. This commonly is used when a series of actions or steps need to be followed in a sequential order.

       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 16–7, 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 16–8.

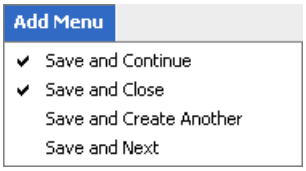
Figure 16–8 Add Menu List



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 16–9.

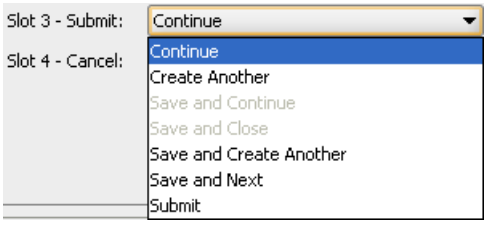
When an item is selected from the Add Menu of Slot 3, the selection of the drop-down in Slot 3 will become the label of `af:commandToolbarButton` and the selections in the Add Menu will become the `af:commandMenuItem` under af:menu in the popup facet of the `af:commandToolbarButton`. The `af:commandToolbarButton` will be added to the `customSaveDropButton facet` (see Table 16–2).

Figure 16–9 Add Menu List Showing Multiple Selections



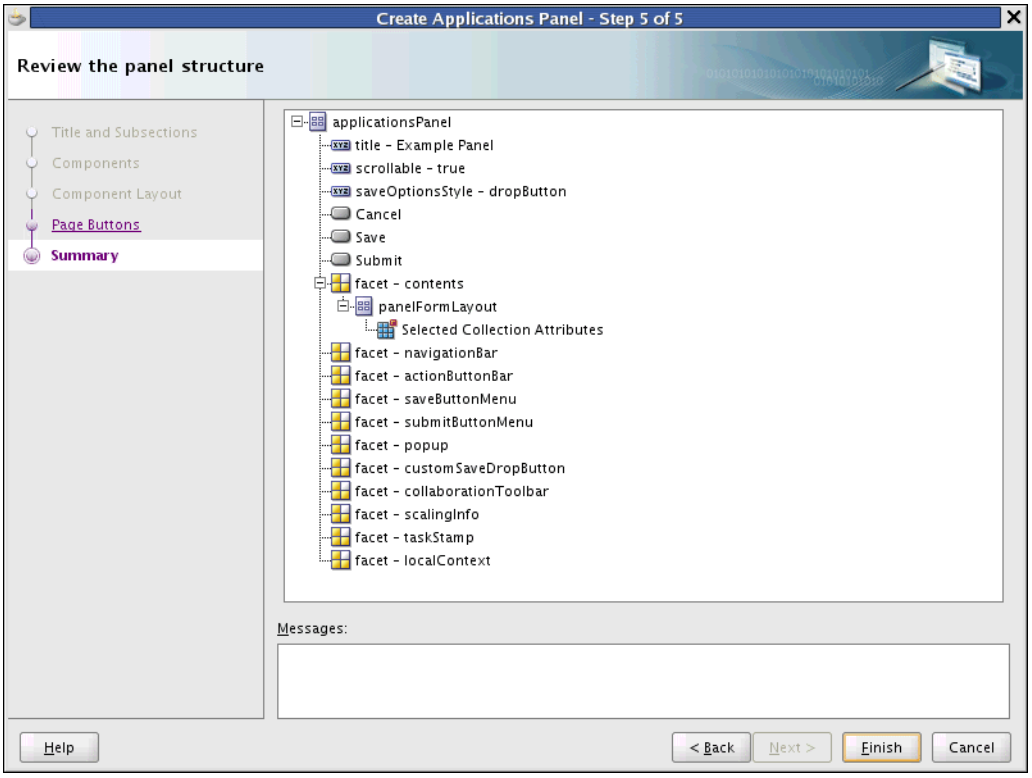
If an option is chosen in the Add Menu of Slot 2, it will be grayed-out as an option for Slot 3 to prevent you from making the same choice multiple times, as shown in Figure 16–10.

Figure 16–10 Add Menu List Selection Effect on Slot 3 Choices



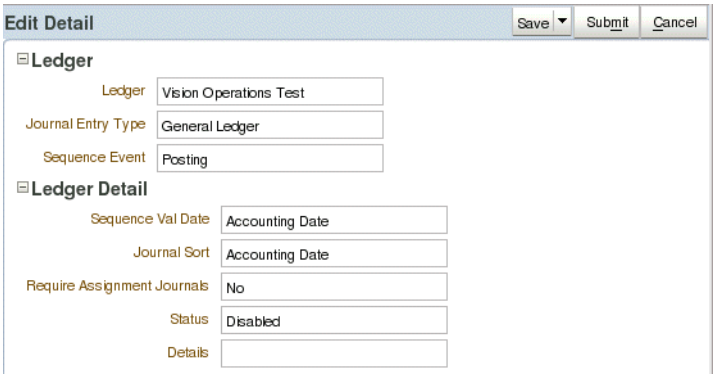
8. Click **Next**. The Summary dialog is displayed, as shown in Figure 16–11.

Figure 16–11 Reviewing the Panel Structure Dialog



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 16–12.

Figure 16–12 Example Page Running In Browser



16.1.3 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.

16.1.3.1 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 "Organizing Content on Web Pages" and "Customizing the Appearance Using Styles and Skins" in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

16.1.3.2 Accessing 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.

16.1.3.3 Editing 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 Section 16.1.3.2, "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 Section 16.1.3.2, "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.

16.1.3.4 Adding 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:

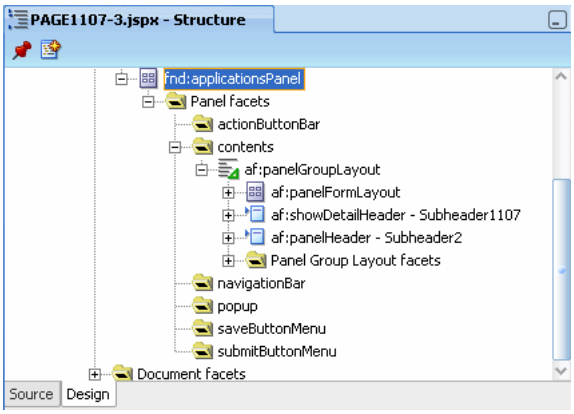
1. In the Application Navigator, open the Data Control panel.
2. Navigate to the data source to bind to the Applications panel.

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 16–13 shows a panel's Structure view, which contains added components.

Figure 16–13 Panel Structure View



   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**.

16.1.3.5 Adding 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 16–2, " Facets of Standard Panel Buttons".

16.2 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 Fusion Developer's Guide for Oracle Application Development Framework*.

16.2.1 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 16–3 shows the facets that are exposed on the Applications Master-Detail.

Table 16–3 Applications Master-Detail Facets

Facet	Description	Allowed Children
`Master`	facet for holding the Applications table	formLayout and Applications Table
`Detail`	facet for holding the Applications Table	Applications Table

View Properties

See Table 15–2, " Applications Table Properties" in Section 15.1, "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 Section 15.1, "Implementing Applications Tables."

16.2.2 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.

16.2.3 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 Web User Interface Developer's Guide for Oracle Application Development Framework*.

Be sure to save your work after you create each component.

16.2.3.1 Adding a Master-Detail to JSF Pages or Page Fragments

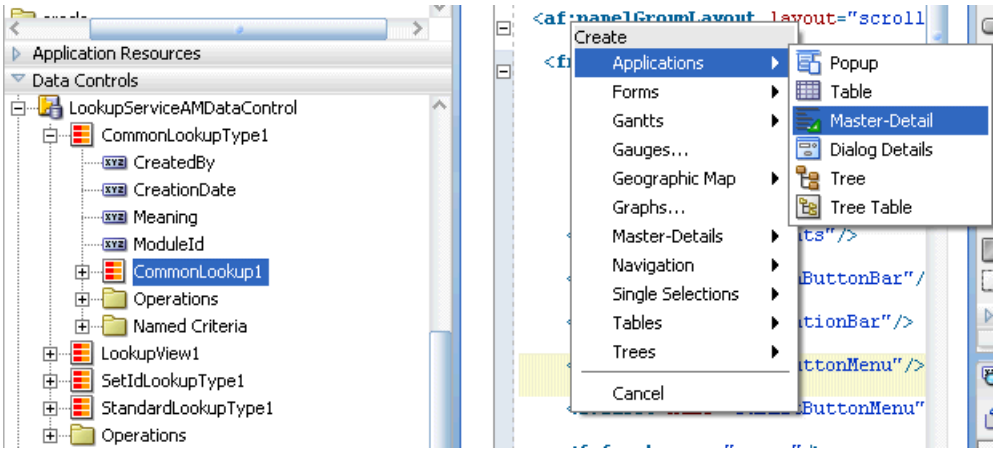
You create a Master-Detail in the Applications Master-Detail wizard, which is displayed when you add the details to your JSF pages (or page fragments) from the **Data Controls** panel.

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 16–14 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 16–14 Example of Master-Detail Relationships



The Applications Master-Detail wizard is displayed.

16.2.3.2 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 once you have enough information to default the rest of the steps. For example, in almost all

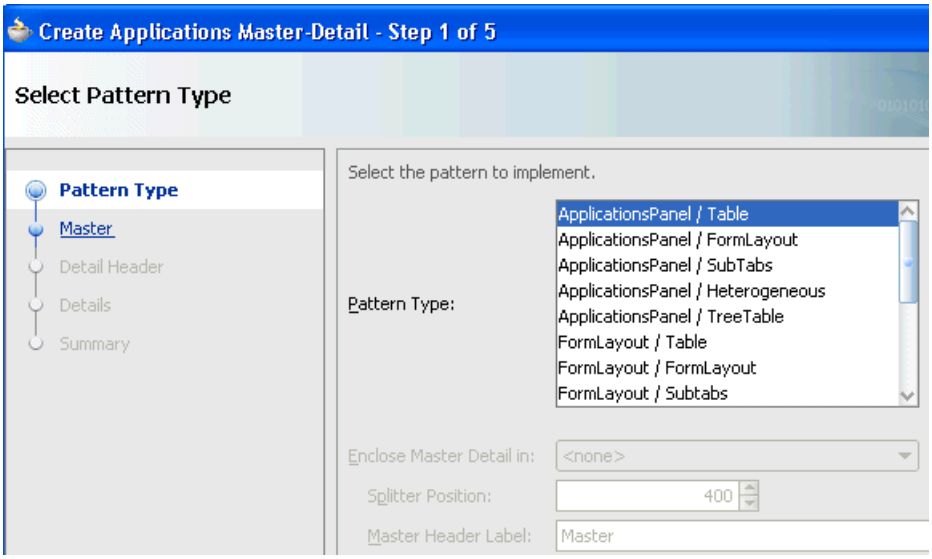
wizards, the last step is the Summary. This step often is omitted, so there is almost always a **Finish** button on the step *before* the Summary step.

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 16–15.

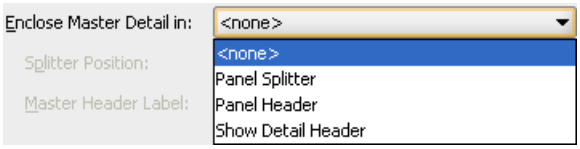
Figure 16–15 Select Pattern Type Dialog



Enclose Master Detail in

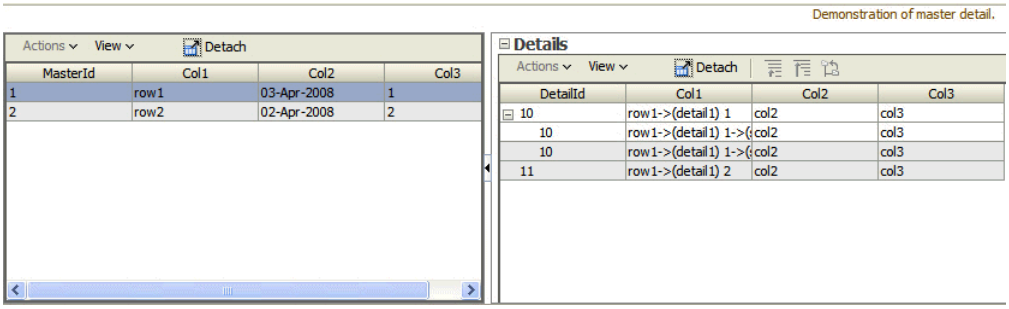
If the Pattern Type supports either the Panel Splitter or the Master Header Label, this option becomes active and a list, shown in Figure 16–16,offers these choices.

Figure 16–16 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 16–17 shows an example of a Splitter in use at the default position.

Figure 16–17 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 16–18, 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 16–18.

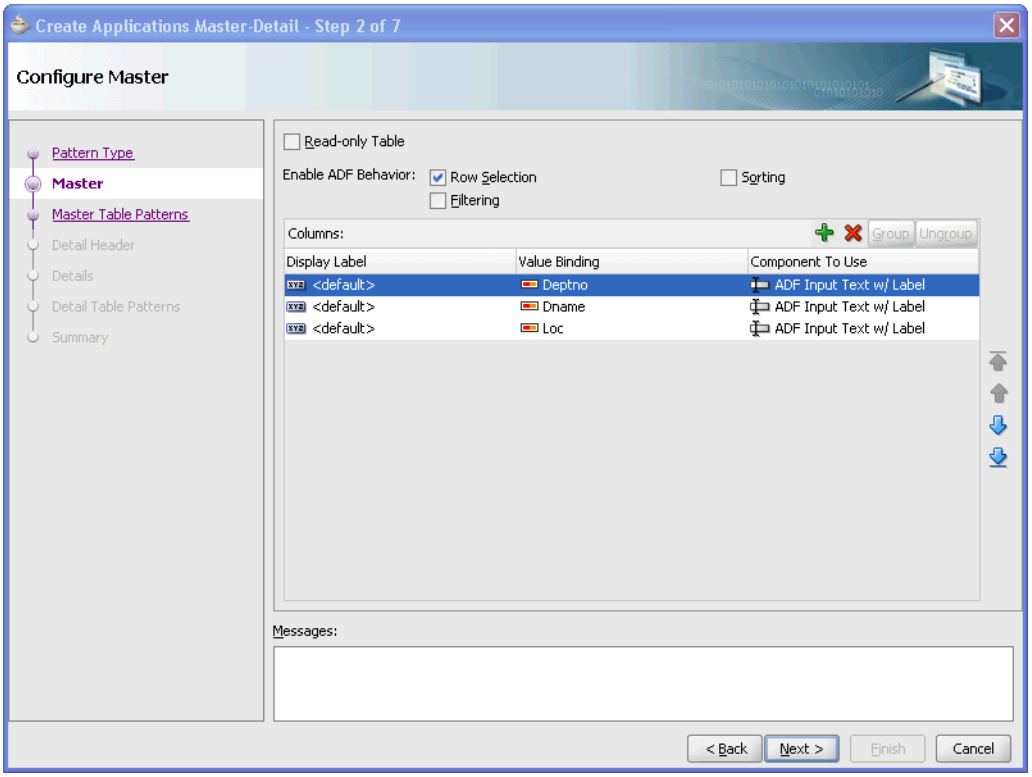
Figure 16–18 Example of Panel and Detail Headers

- **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 16–19.

Figure 16–19 Table/Table Configure Master Dialog

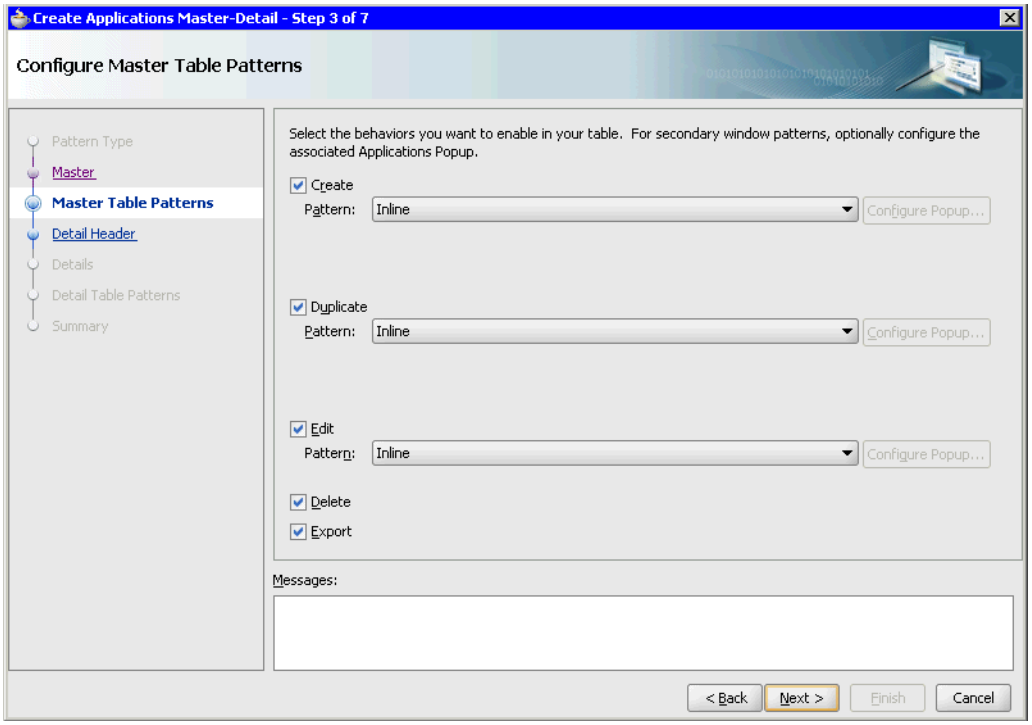


To create your Master-Detail for your Master table columns:

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.

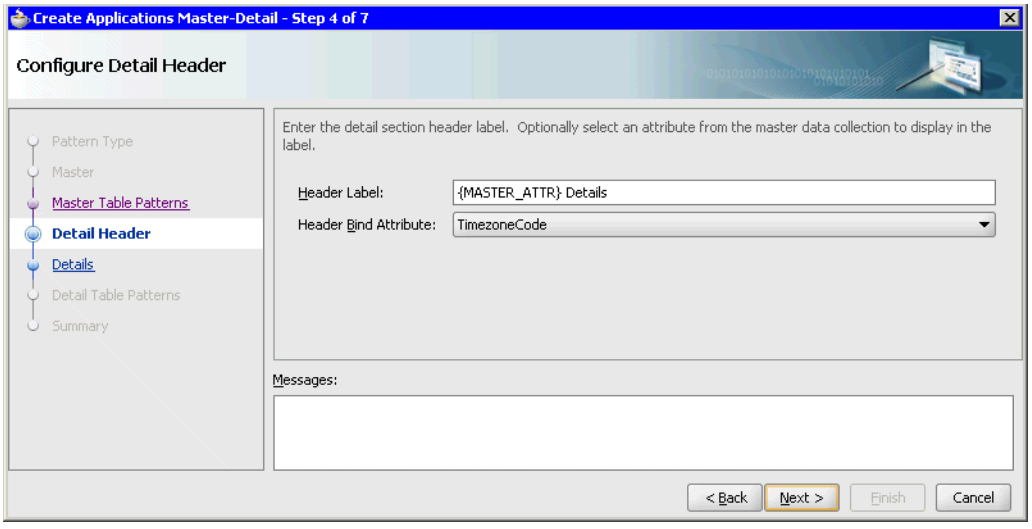
   Click **Next** to display the Configure Master Table Patterns dialog, shown in Figure 16–20.

Figure 16–20 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 16–21.

Figure 16–21 Configure Detail Header Dialog



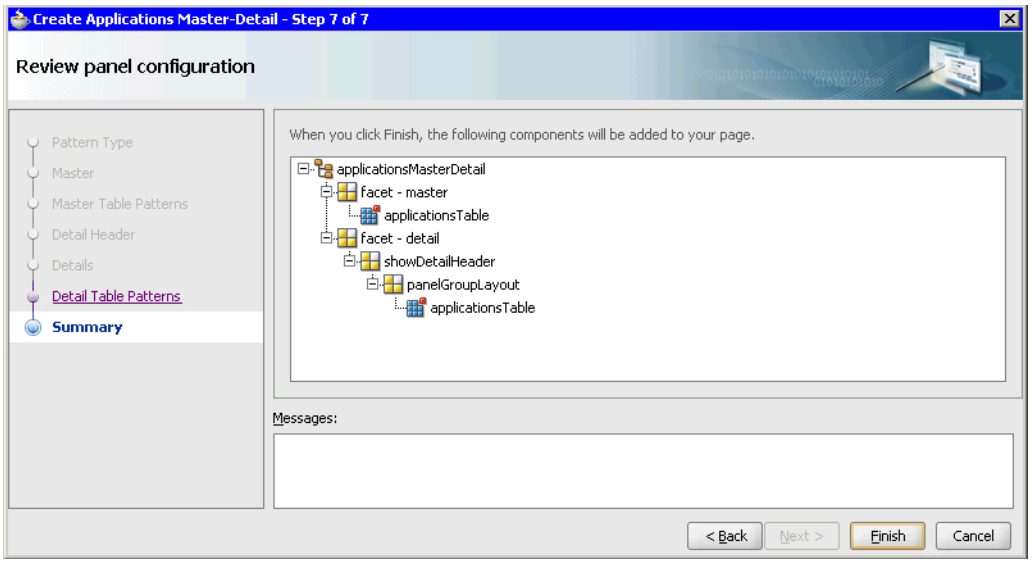
6. Enter a Detail header name and choose a corresponding attribute.
7. Click **Next** to display the Configure Details dialog:

   Configuring the Details table is the same as configuring the Master table; see Figure 16–19 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 16–20 and step 5.

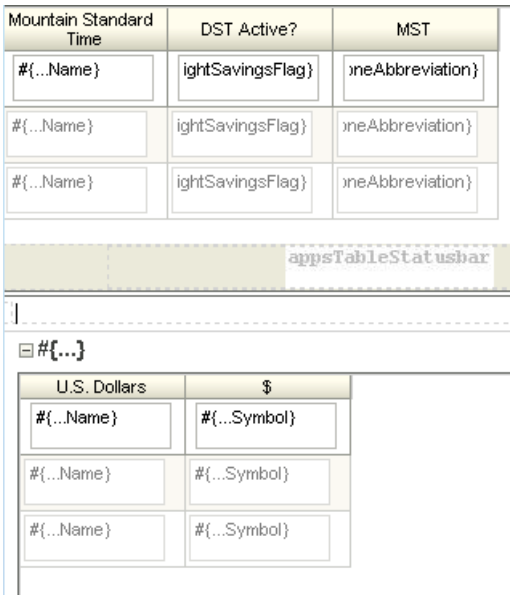
9. Click **Next** to display the Review panel configuration dialog, shown in Figure 16–22.

Figure 16–22 Reviewing the Panel Configuration



When you click **Finish**, the Table/Table Master-Detail is added to the editor, and appears similar to Figure 16–23 in Design mode.

Figure 16–23 Table/Table Master-Detail Example



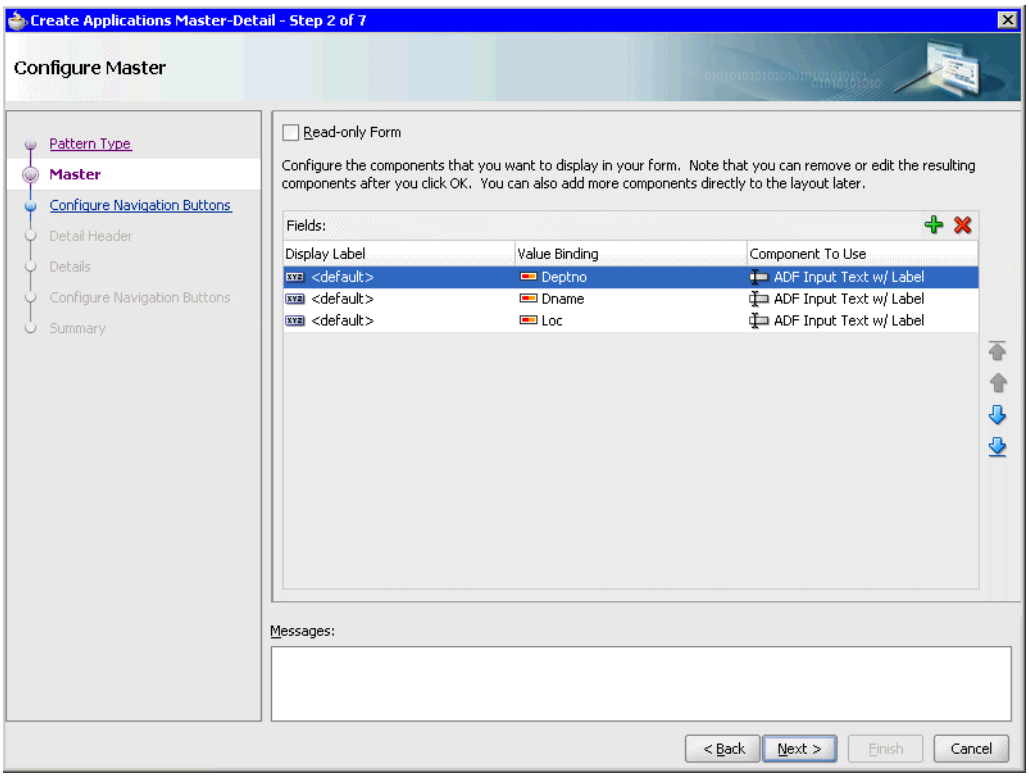
Creating a Master-Detail Using Forms

When the Create Applications Master-Detail wizard launches, the Select Pattern Type dialog displays, shown in Figure 16–15.

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 16–24.

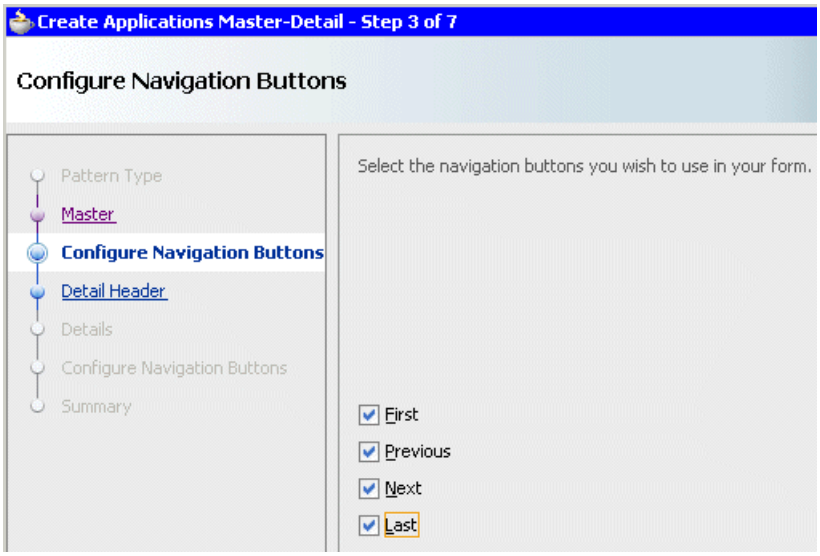
Figure 16–24 FormLayout /FormLayout Configure Master Dialog



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 16–19.

2. Click **Next** to display the Configure Navigation Buttons dialog, shown in Figure 16–25.

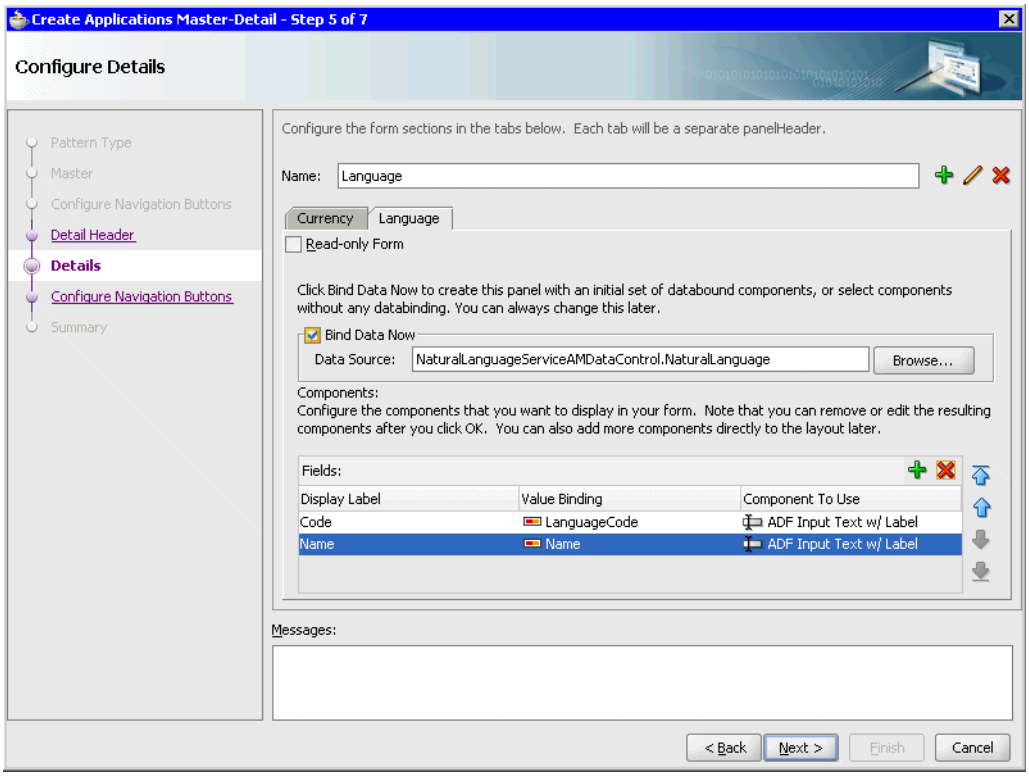
Figure 16–25 Form/Form Configure Navigation Buttons Dialog



Select the navigation buttons you want to appear on the Main form.

3. Click **Next** to display the Detail Header dialog, shown in Figure 16–21 for details.
4. Click **Next** to display the Configure Details dialog, shown in Figure 16–26.

Figure 16–26 Form/Form Configure Details Dialog



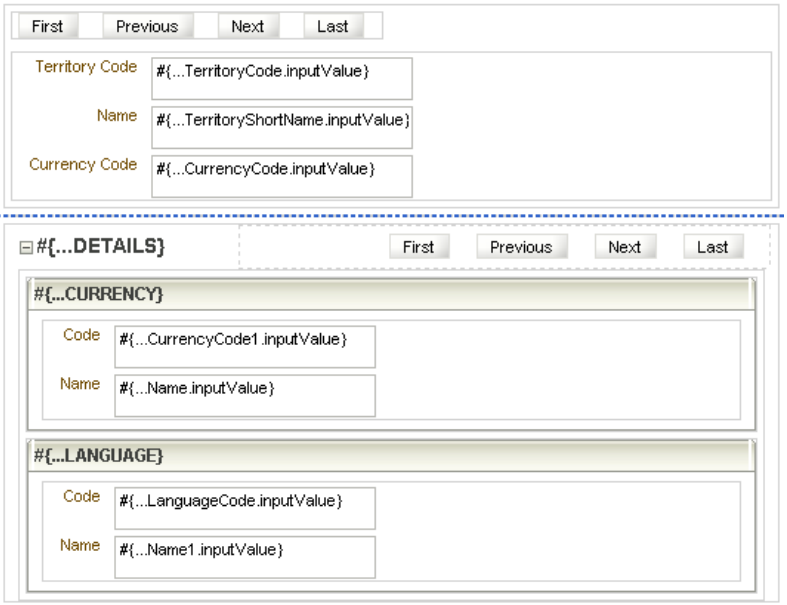
Use this dialog to create as many tabs on the Details form as you need. To create a new tab, enter a name in the Name field and click the Add icon. The tab is added in the area beneath the Name field.

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 16–19.

5. Click **Next** to display the Configure Navigation Buttons for the Detail section dialog, shown in Figure 16–25.
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 16–27.

Figure 16–27 Form/Form Master-Detail in Page Editor



16.2.4 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.

16.2.4.1 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 needs to be clicked to save the master data before creating detail data.

16.2.4.2 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`.

16.2.4.3 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.

16.2.4.4 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.

16.2.5 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.

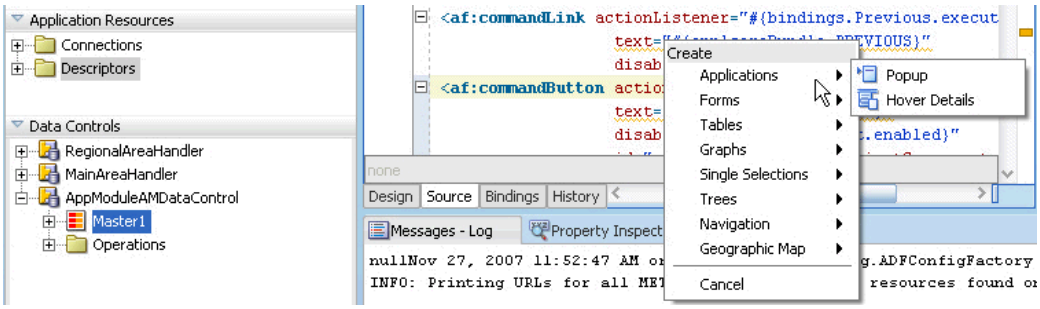
16.3 Implementing Hover

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 16–28.

Figure 16–28 Dragging from the Component Palette onto a Drop Component



The allowed drop components are:

- `af:commandLink`
- `af:commandImageLink`
- `af:commandToolbarButton`

The DT will check to see whether the component is in a table already bound, and if that binding is for the collection being dropped.

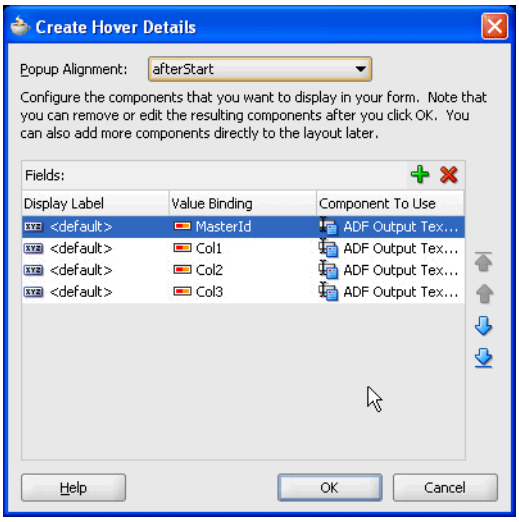
- If so, it will create additional table bindings for the attributes selected.
- If not, it will create form bindings for the collection.

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 16–29.

Figure 16–29 Hover Popup Attributes



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:

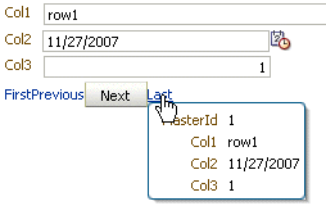
- The drop component will be given an Id if it does not have one already, and have the clientComponent attribute set to true.
- The drop component will have a `<af:showPopupBehavior>` component added as a child.
- A popup (`<af:popup>`) will be added as the previous sibling of the drop component, shown in Example 16–2 for the sample markup, and Figure 16–30 for a sample of the result.

Example 16–2 Example Markup for a Form-based Layout

```
<af:popup id="popup1">
 <af:panelFormLayout labelAlignment="start">
   <af:panelLabelAndMessage label="#{bindings.MasterId.hints.label}">
     <af:outputText value="#{bindings.MasterId.inputValue}">
       <af:convertNumber pattern="#{applCorePrefs.numberFormatPattern}"/>
     </af:outputText>
   </af:panelLabelAndMessage>
   <af:panelLabelAndMessage label="#{bindings.Col1.hints.label}">
     <af:outputText value="#{bindings.Col1.inputValue}"/>
   </af:panelLabelAndMessage>
```

```
   <af:panelLabelAndMessage label="#{bindings.Col2.hints.label}">
     <af:outputText value="#{bindings.Col2.inputValue}">
       <af:convertDateTime pattern="#{applCorePrefs.dateFormatPattern}"/>
     </af:outputText>
   </af:panelLabelAndMessage>
   <af:panelLabelAndMessage label="#{bindings.Col3.hints.label}">
     <af:outputText value="#{bindings.Col3.inputValue}">
       <af:convertNumber pattern="#{applCorePrefs.numberFormatPattern}"/>
     </af:outputText>
   </af:panelLabelAndMessage>
 </af:panelFormLayout>
</af:popup>
<af:commandLink actionListener="#{bindings.Last.execute}"
  text="#{applcoreBundle.LAST}"
  disabled="#{!bindings.Last.enabled}"
      id="rolloverComponent2" clientComponent="true">
 <af:showPopupBehavior triggerType="mouseOver" popupId="popup1"
        alignId="rolloverComponent2"
        align="afterStart"/>
</af:commandLink>
```

Figure 16–30 Example of a Form-based Layout



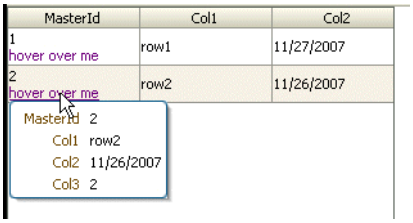
Example 16–3 shows the sample markup for a table-based layout and Figure 16–31 shows an example of how the result appears.

Example 16–3 Example Markup for a Table-based Layout

```
<af:table value="#{bindings.Master1.collectionModel}" var="row"
                  rows="#{bindings.Master1.rangeSize}"
                  emptyText="#{bindings.Master1.viewable ? applcoreBundle.TABLE_
EMPTY_TEXT_NO_ROWS_YET : applcoreBundle.TABLE_EMPTY_TEXT_ACCESS_DENIED}"
                  fetchSize="#{bindings.Master1.rangeSize}">
          <af:column sortProperty="MasterId" sortable="false"
                     headerText="#{bindings.Master1.hints.MasterId.label}">
                <af:outputText value="#{row.MasterId}">
                  <af:convertNumber
pattern="#{applCorePrefs.numberFormatPattern}"/>
                </af:outputText>
            <af:popup id="popup1">
              <af:panelFormLayout>
                <af:panelLabelAndMessage
label="#{bindings.Master1.hints.MasterId.label}">
                  <af:outputText value="#{row.MasterId}">
                    <af:convertNumber
pattern="#{applCorePrefs.numberFormatPattern}"/>
                  </af:outputText>
                </af:panelLabelAndMessage>
                <af:panelLabelAndMessage
label="#{bindings.Master1.hints.Col1.label}">
                  <af:outputText value="#{row.Col1}"/>
                </af:panelLabelAndMessage>
```

```
                <af:panelLabelAndMessage
label="#{bindings.Master1.hints.Col2.label}">
                  <af:outputText value="#{row.Col2}">
                    <af:convertDateTime
pattern="#{applCorePrefs.dateFormatPattern}"/>
                  </af:outputText>
                </af:panelLabelAndMessage>
                <af:panelLabelAndMessage
label="#{bindings.Master1.hints.Col3.label}">
                  <af:outputText value="#{row.Col3}">
                    <af:convertNumber
pattern="#{applCorePrefs.numberFormatPattern}"/>
                  </af:outputText>
                </af:panelLabelAndMessage>
              </af:panelFormLayout>
            </af:popup>
            <af:commandLink ="hover over me" id="rolloverComponent2"
                            clientComponent="true">
              <af:showPopupBehavior triggerType="mouseOver" popupId="popup1"
                                    alignId="rolloverComponent2"
                                    align="afterStart"/>
            </af:commandLink>

          </af:column>
```

Figure 16–31 Example of a Table-based Layout



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.

16.4 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.

16.4.1 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 16–4 shows the properties that are exposed on the Applications Dialog Details.

Table 16–4 Applications Dialog Details Properties

Property	Description	Allowed Values
`Id (id)`	The unique ID for this Applications Table	string
`Rendered (rendered)`	Whether the Applications Table is rendered or not	boolean / expression
`Detail Pattern Type (detailPatternType)`	Detail pattern type	image, link or button
`Popup Id (popupId)`	ID of the popup to be invoked when Detail image/link/button is clicked	string
`Text (text)`	Overrides default label for Detail button, or defines link text for Detail link	expression
`Short Description (shortDesc)`	Overrides default roll-over text for detail image/button	expression
`Disabled (disabled)`	Sets whether the component needs to be disabled	boolean / expression

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.

16.4.1.1 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. Once you complete this wizard, you can further refine the dialog details definition by editing the resulting component as needed.

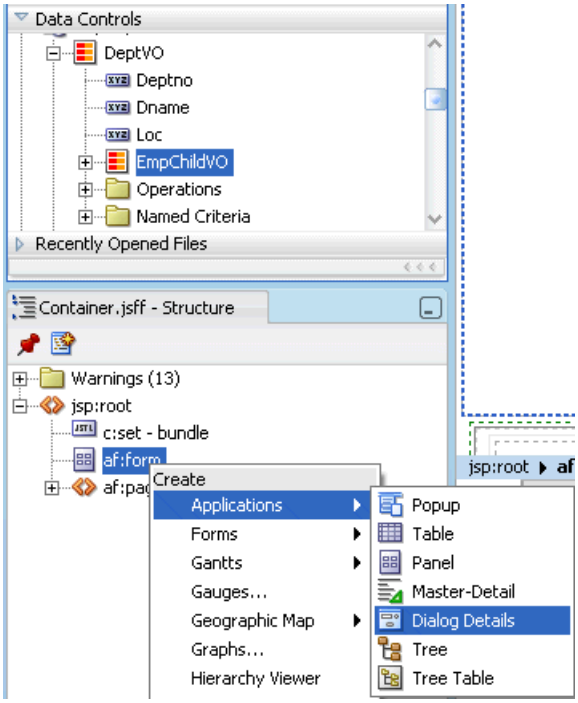
Component First

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 16–32.

Figure 16–32 Launch the Applications Dialog Detail Wizard



Applications Dialog Details Create Wizard

The Applications Dialog Details Create wizard consists of only one panel.

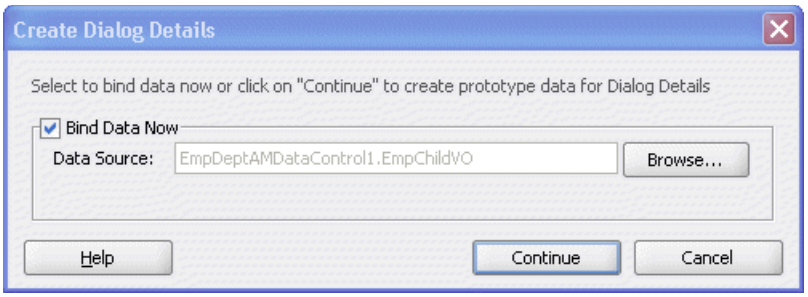
Create Applications Dialog Details Panel

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 16–33.

Figure 16–33 Create Applications Dialog Details Panel



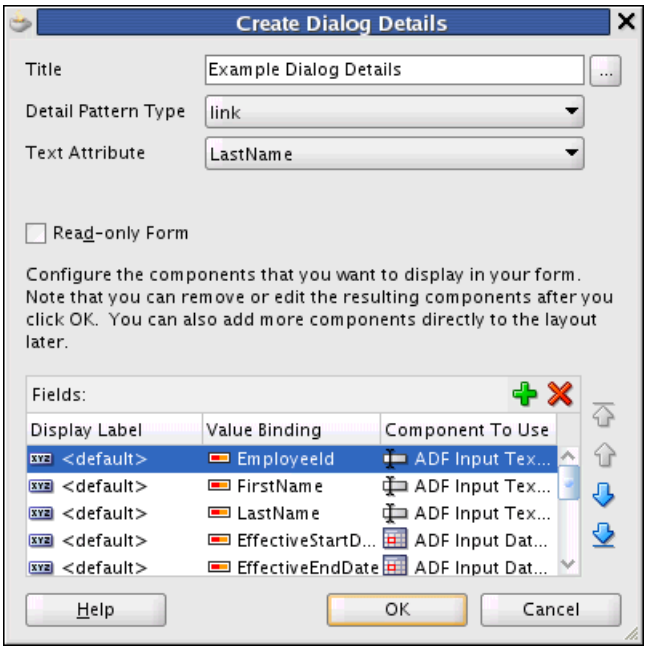
You can skip the data control binding step when creating the Applications Dialog Details. In this case, the Applications Dialog Details will create several default placeholder `outputText` fields that you can use for layout purposes in the popup. You can decide how many placeholder columns you wish to display. Once 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**.

Once 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 16–34.

Figure 16–34 Select a Data Source Attribute



Title

- If the dialog will be read-only:

  The format should be `<Object Type> <Object Name>` (such as Expense Report WBJ3008D)

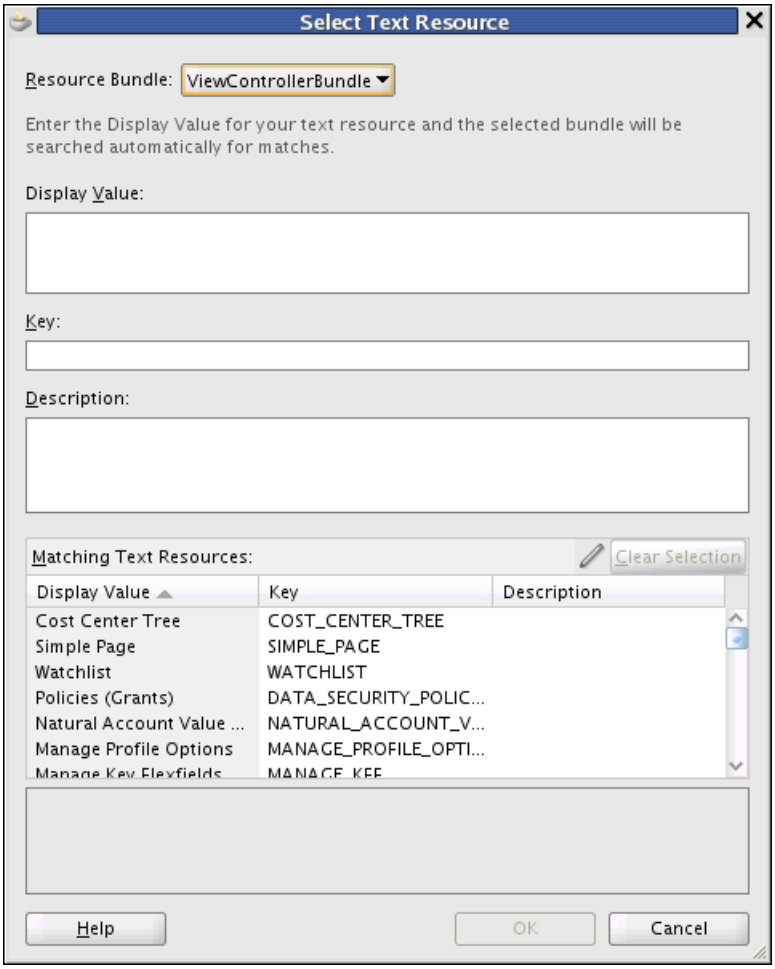
- If the dialog contains editable fields:

  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 16–35.

Figure 16–35 Select Text Resource for Dialog Details Title

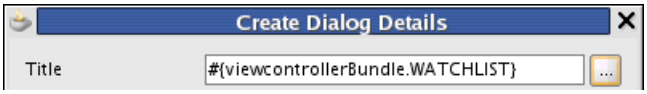


- **Resource Bundle**: Select the bundle containing the string you want to use. You can select a single bundle or all available bundles. The strings will display in the Matching Text Resources field.
- **Display Value**: You can enter a new string for a title here.
- **Key**: Each resource must have 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 16–36.

Figure 16–36 Dialog Details Title Field Using Resource Bundle



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

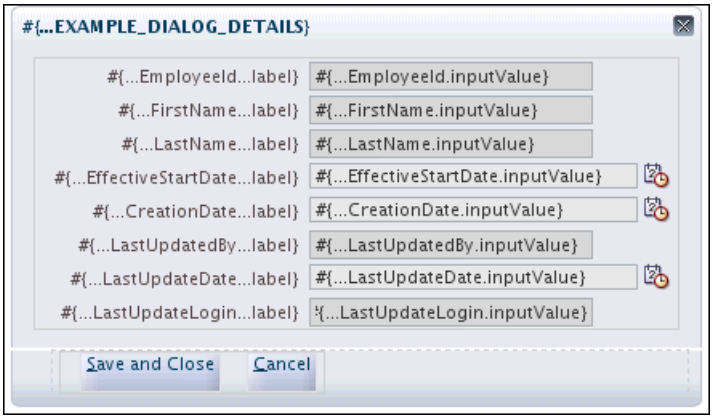
This setting is available only if the Detail Pattern Type is **link**. The text entered here is shown as the Dialog Details link. This helps give the user an idea about the data contained in the popup.

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 16–37 shows the default buttons in the form in the JDeveloper Design view.

Figure 16–37 Default Buttons Added to non Read-only Form



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 16–38.

Figure 16–38 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 or not you choose read-only. If you selected Read-only, the choices will change from Input Text to Output Text types.

16.4.1.2 Working with the Applications Dialog Details

This section discusses how to edit Dialog Details properties.

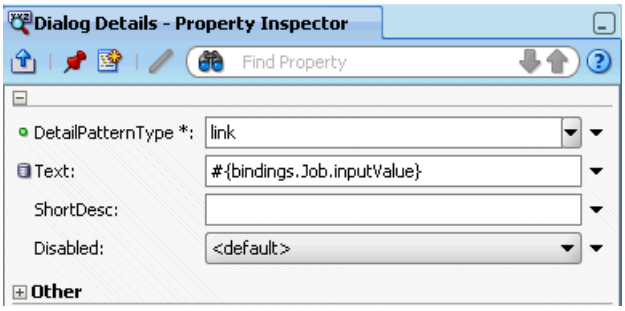
Editing - Properties

Once 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 16–39.

Figure 16–39 Dialog Details Property Inspector



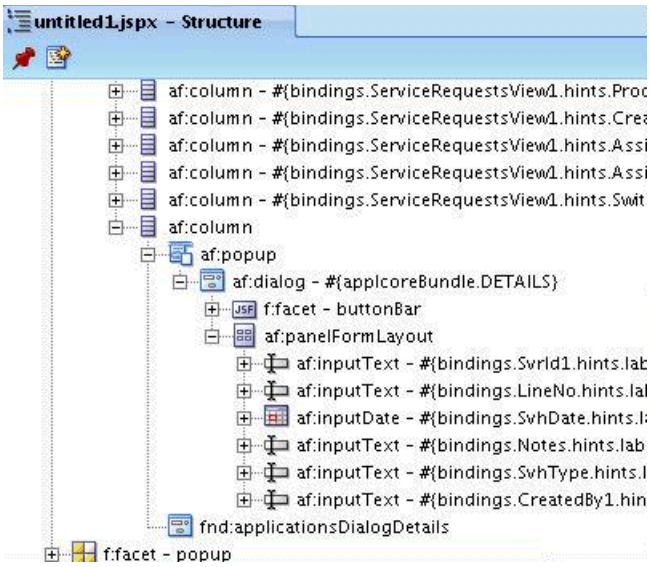
Adding a Data Source

Once 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 16–40 shows the Structure view of a page with components already added.

Figure 16–40 Page Structure View



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`.

16.4.1.3 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. Once the buttons have been added, create a managed bean to set each button's action listener. Use the method in Example 16–4 as the **Cancel** button's action listener.

Example 16–4 Example Method to Create a Managed Bean to Be the Cancel Button's Action Listener

```
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);
  }
```

Note: Although the `closePopup()` implementation shown in Example 16–4 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 or not you have the taskflow return activity.

```
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 16–4, 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 16–5.

Example 16–5 Example Method to Create a Managed Bean to Be the OK Button's Action Listener

```
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);

            }
        }
```

17 Implementing Skinning

This chapter describes how to change the look and feel of your application by changing the skin. This chapter deals specifically with skinning as applied to Oracle Fusion applications and the UI Shell.

This chapter includes the following sections:

- Section 17.1, "Implementing Skinning"
- Section 17.2, "Creating and Implementing a Custom Skin"
- Section 17.3, "Changing the Skin of an Application"

For general information about skinning, see "Customizing the Appearance Using Styles and Skins" in the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

For information about the UI Shell, see Chapter 14, "Implementing the UI Shell."

17.1 Implementing Skinning

Skinning lets you change the look and feel of your application without changing the content.

The goals of skinning include:

- Allow business managers to update branding rather than a software developer.
- Allow all Oracle Fusion applications to be updated through a single source of information.
- Allow different branding for partners or groups of partners accessing the same application.
- Generate a nice look and feel, such as rounded corners and color gradations, rather than a flat box-like look, unless that is desired.
- Keep the process manageable and keep the rebranding through future patches and upgrades.

17.1.1 Before You Begin

Before you can implement skinning, you need to accomplish these actions:

- Install the ADF Skin Editor. Follow the directions in the *Oracle Fusion Middleware Installation Guide for Oracle Application Development Framework Skin Editor*.
- Launch the skin editor. See the *Oracle Fusion Middleware Skin Editor User's Guide for Oracle Application Development Framework*.

The first time you run, you may be prompted to locate the path to your Java executable:

```
Type the full pathname of a J2SE installation (or Ctrl-C to quit), the path
will be stored in ~/.skineditor_jdk
```

Note: If the Java executable is in the `bin` subdirectory, do not add `bin` to the path you enter. `/bin` is added automatically to the end of the string you enter.

- Check for updates. Select **Help > Check for Updates**.

17.2 Creating and Implementing a Custom Skin

Follow these basic steps to create a custom skin and implement it in your application.

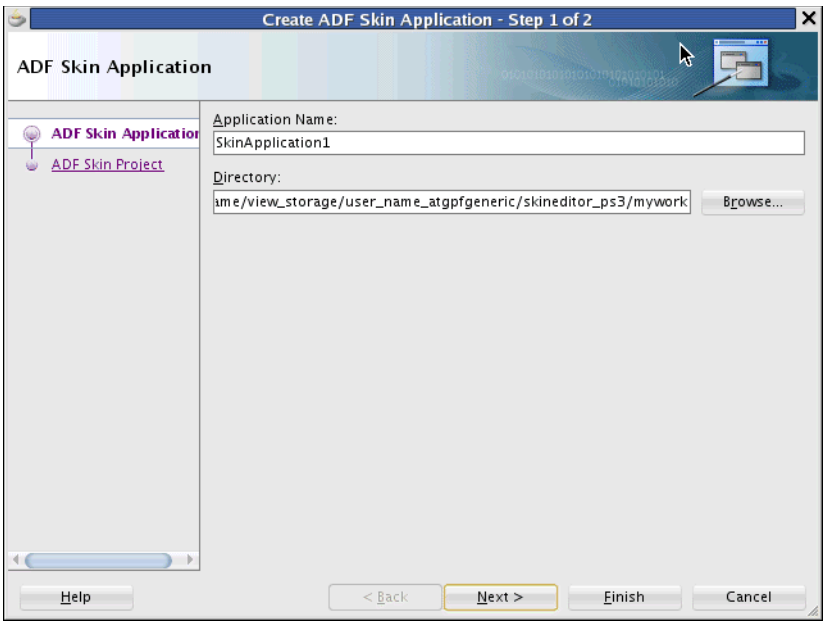
- Create a skin using the ADF Skin Editor. See Section 17.2.1, "How to Create the Skin Project."
- Add the skin to your application. See Section 17.3, "Changing the Skin of an Application."
- Set the FND_CSS_SKIN_FAMILY profile option.
- Restart your application.

17.2.1 How to Create the Skin Project

The first step to create a custom skin is to create the skin project. Follow these steps.

1. Launch the ADF Skin Editor.
2. Create a new application. Select **File > New > ADF Skin Application**. The Create ADF Skin Application wizard launches and displays the ADF Skin Application dialog, as shown in Figure 17–1.

Figure 17–1 Naming the Skin Application



- **Application Name**

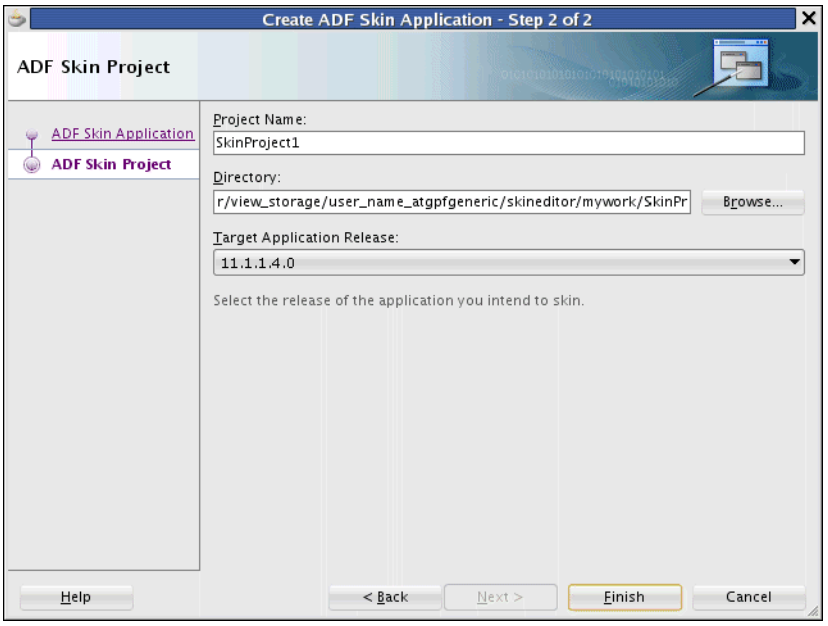
Enter a name for the application. By default, an application name in the form **Application*n*** is shown, where *n* is a number that increases sequentially from 1. This is the filename that will be used for the application control file within the file system. The extension **.jws** is assumed, but not displayed.

- **Directory**

  Enter a directory for the application or click **Browse** to locate one. The default directory location is its own subdirectory beneath **skineditor/**, such as **skineditor/Application_Name**.

Click **Next** to display the ADF Skin Project dialog, shown in Figure 17–2.

Figure 17–2 Naming the Skin Project



- **Project Name**

  Enter a name for the project. By default, a project name in the form **Project*n*** is shown, where *n* is a number that increases sequentially from 1. This is the filename that will be used for the project within the file system. The extension **.jpr** is assumed, but not displayed.

- **Directory**

  Enter a directory for the application or click **Browse** to locate one. The default directory location is **skineditor/Application_Name/Project_Name**.

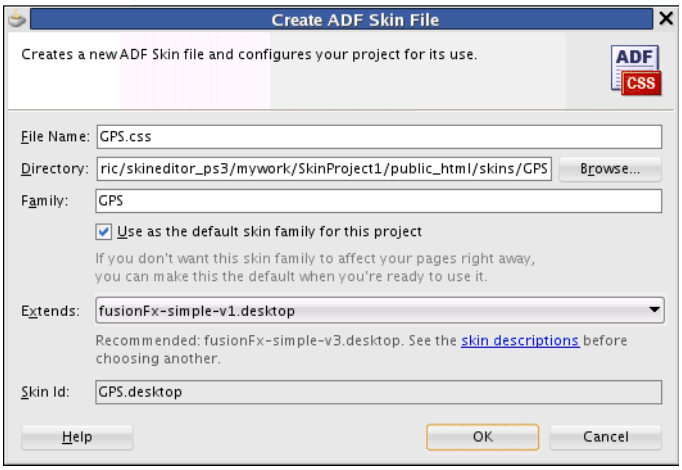
- **Target Application Release**

  Choose 11.1.1.5.0, or whatever is the Fusion Applications release you are running (choose Help > About to see the release number). This determines which skin families are available for your release and the values that will be available for the **Extends** option when you create a new ADF Skin file, as shown in Figure 17–3.

Click **Finish**.

3. Select **File > New > ADF Skin File** to launch the Create ADF Skin File wizard, shown in Figure 17–3.

Figure 17–3 Creating a Skin File



An ADF skin is a type of CSS file that you can use to define the look and feel of your application. Oracle Application Development Framework (ADF) provides a number of ADF skins that you can extend when you create a new ADF skin. The recommended ADF skin to extend depends on the release of Oracle ADF that you use.

- **File Name**

  Enter a file name for the new ADF skin. The example uses GPS.

- **Directory**

  Enter the directory path to the CSS source file for the ADF skin. You can accept the default path shown in the Directory field specify a different path.

- **Family**

  This value automatically is populated with the root of the File Name you entered.

  If necessary, enter a different value for the family name of your ADF skin. Later, you will set this value in your application's configuration file (for example, the **trinidad-config.xml** file) to apply the ADF skin to the application.

  – **Use as the default skin family for this project**

    Clear this checkbox if you do not want to make the ADF skin the default for your project. Later, you can set a value in your project's configuration file to make the ADF skin the default.

- **Extends**

  This field lets you select from the list of available skin families. The selection depends on the value selected for the **Target Application Release**, as shown in Figure 17–2.

  From the drop-down list, select **fusionFx-simple-v1.1.desktop**. Newer releases of software may use different styles. See release notes.

  Compared to the complex Oracle-specific skin, this skin is optimized to make it easy to create a new style.

  – Simplified color selection

  – Better aliases

– Images that can be colored automatically

The difference between extending and adding is that, by adding, you are still referencing the base skin family (such as Fusion-Fx in the Trinidad-config file) but the styles in the addition also are picked up. Extending is when you extend and then override.

- **Skin ID**

  This read-only value automatically is populated with the root of the **File Name** you entered and the extension of the value selected in the **Extends** field.

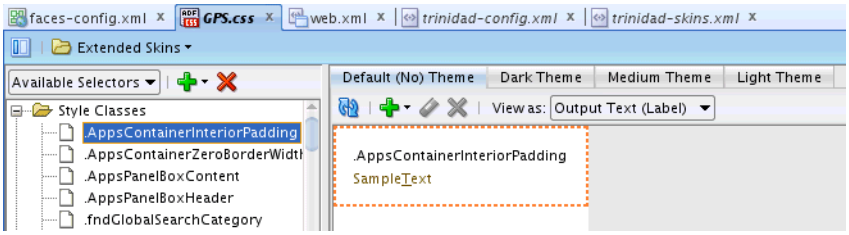
  The **trinidad-skins.xml** file uses the value of Skin Id to identify ADF skins.

Click **OK** to create the .css file (GPS.css in this example).

17.2.2 How to Customize the Skin

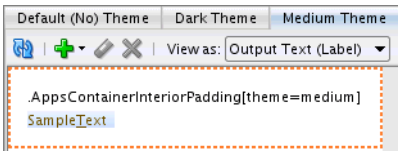
When you click to display the Design view, the styles display in the Style Classes list and a sample view is displayed, as shown inFigure 17–4.

Figure 17–4 Showing Style Classes in a Skin CSS File



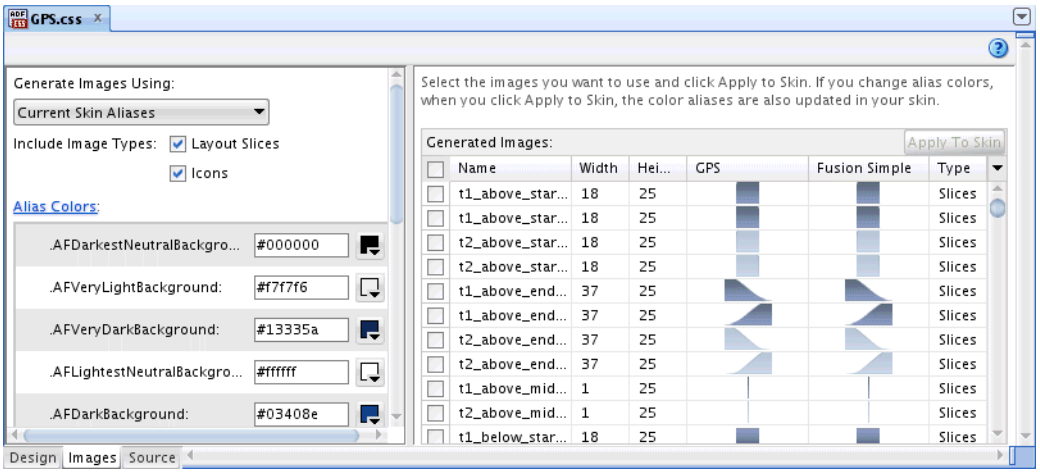
Click another Theme tab, such as Medium Theme, to show how the text display changes, as shown in Figure 17–5.

Figure 17–5 Displaying a Different Theme



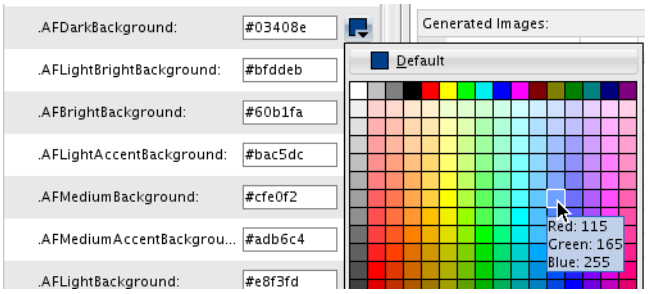
To customize the colors of the CSS style, click the Images tab. The display will change to resemble Figure 17–6.

Figure 17–6 Displaying a Scheme's Colors



To change a style's color, click the icon to the right of the style name to display the color picker, shown in Figure 17–7.

Figure 17–7 Displaying the Color Picker



17.2.3 How to Deploy the Skin Profile

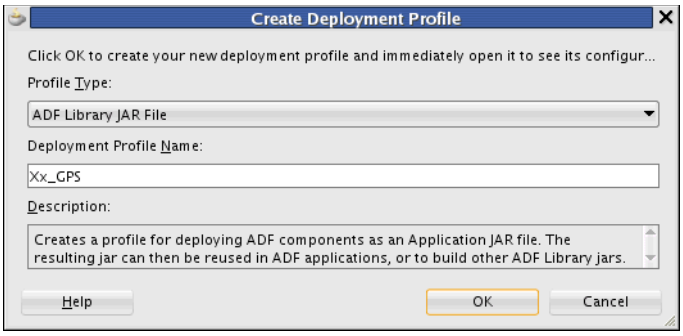
You need to deploy the skin project to a JAR file that can be included in an application. There are three steps to deploying the skin project:

- Create the skin deployment profile.
- Deploy the skin profile.
- Deploy the skin profile to a JAR file.

To deploy the skin project:

- Right-click the skin project and select **Deploy > New Deployment Profile** to display the Create Deployment Profile dialog, shown in Figure 17–8.

Figure 17–8 Creating the Skin Deployment Profile



  - **Profile Type**

    Select ADF Library JAR File.

  - **Deployment Profile Name**

    The name *must* begin with **Xx_**. The example uses Xx_ plus the name of the project.

- Click **OK**.
- Right-click the skin project and select **Deploy** > ***profile_name*** to display the Deploy *profile_name* dialog.
- Click **Finish**.
- Right-click the skin project and select **Deploy** > ***profile_name*** **to JAR file**.

The skin profile JAR now is ready to be added to an application's Libraries and Classpath.

17.2.4 How to Change the Logo

The logo to be used can be located through a URL where all applications point to a single location, or packaged within a JAR file and placed in the class path of each application. The trade-offs are that setting up a single location is easier, but if the server where the logo is located is down, the logo will not appear. A logo located within each application always will be available for that application.

To create a JAR file with a logo:

- Create a new project in JDeveloper.
- Place the image in the public_html folder.
- Deploy to an ADF Library JAR file with a name starting with "**Xx**."
- Place the new JAR file in the application's exploded WEB-INF/lib directory.
- Change the template to reference the logo using the Edit UIShell template taskflow, or JDeveloper customization.

The reference in the template to the image is just the name of the logo file:

```
<af:image id="oracleImage"
source="myLogo.png"
```

If your logo height is greater than 30 pixels, change the height to fit the logo. To achieve this, change the topHeight in 2 locations from 30px to your logo height. Depending on padding in the skin style, you may need to add a couple pixels to prevent a scroll bar from appearing.

```
<af:panelStretchLayout id="pt_psl1" topHeight="30px"
<af:panelStretchLayout id="_UISpsl1" topHeight="30px"
```

You also need to change the height of the UIShellBrandingBarItemTop:

```
.UIShellBrandingBarItemTop {
     height: 54px; /* Match the logo height if > 30px */
}
```

17.3 Changing the Skin of an Application

You can change the skin of an existing deployed application or you can add a skin within JDeveloper to a development project.

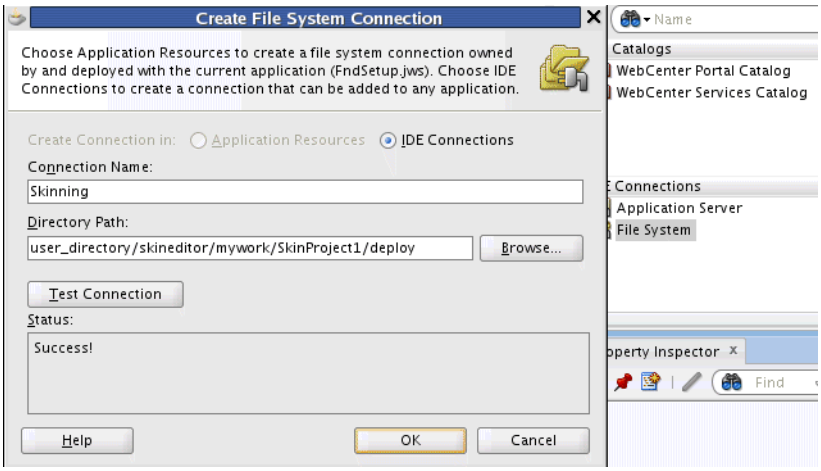
To change the skin of a deployed application:

- Add these three JAR files to your application's `WEB-INF/Lib` directory:
  - Your new skin, Xx_GPS in this example.
  - The **skineditor/jlib/adf-richclient-fusion-simple_11.1.1.5.0.jar** that is located within the skin editor files already downloaded from Oracle Technology Network (OTN).
  - The **Xx_ApplCoreStyles.jar** (located on OTN)
- Stop and restart your application.

To use a skin within JDeveloper:

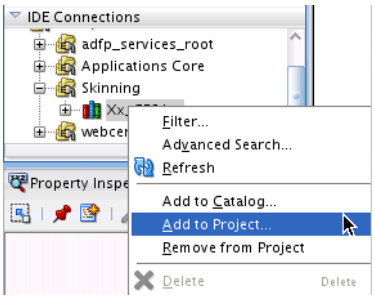
- Add the directory to which the skin JAR file (Xx_GPS in the examples) was deployed, into the Resource pallet, as shown in Figure 17–9.

Figure 17–9 Creating Skinning File System Connection



  - Right-click the library and choose **Add to Project**, as shown in Figure 17–10.

Figure 17–10 Adding Skin Library to JDeveloper Project



  - Click **OK** when prompted to add the library to the View Controller project. It will show in your project as "ADF Library."

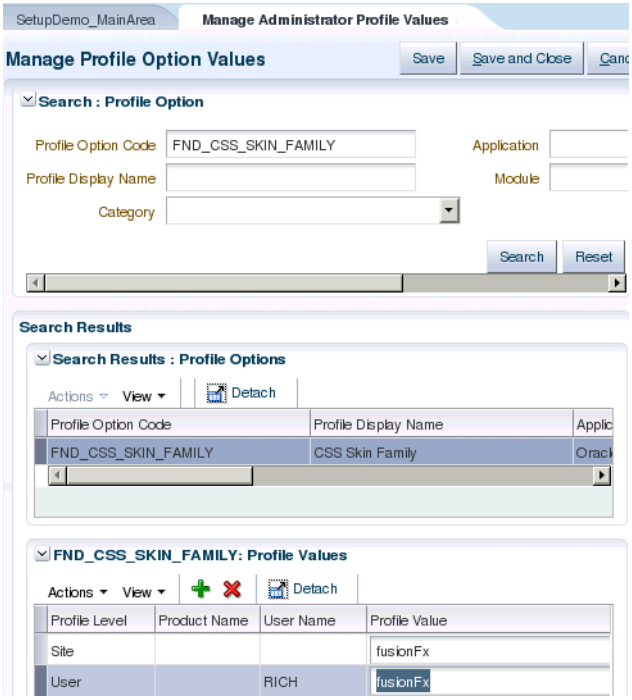
- Redeploy your application.

Set your profile option (PO) to the new skin. (You can create a PO for the skin associated to a single user when developing a skin so others are not affected.) After changing the PO, reload a page using the PO and you should see your changes immediately.

To change the PO:

- Deploy the **fndSetup** application.
- In the Tasklist, select **Profiles > Manage Administrator Profile Values**.
- In the Search: Profile Option section, enter FND_CSS_SKIN_FAMILY (or FND_CSS%) in the Profile Option Code field and click **Search**. The complete Profile Option values dialog is shown in Figure 17–11.

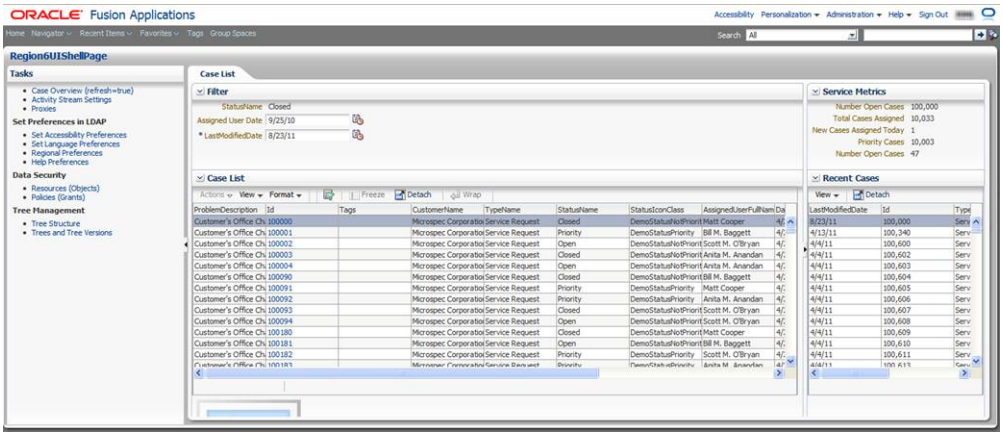
Figure 17–11 Changing the Skin Using Profile Option



- Change the value to the new skin family your skin is using. For instance, if you were using the name RICH, you would change the Profile Value from fusionFx to Xx_GPS.
- Click **Save**. A simple change in the skin profile option does not require you to restart the server or redeploy the application, but it does require the user to log off and back on to see the new skin.

If you log out of the fndSetup application and then back in, you'll see that the simple Xx_GPS skin has changed how the application looks, as shown in Figure 17–12.

Figure 17–12 Showing the Effect of the New Skin



18 Implementing Attachments

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:

For general information about Oracle Fusion Middleware user interface (UI) components, see:

- "Introduction to Oracle ADF Applications" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- "Introduction to ADF Faces Rich Client" chapter in the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*

18.1 Introduction to Attachments

The Attachment component provides a declarative programming mechanism for you to add attachments to the user interface (UI) pages that you create for Fusion web applications. Once 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. 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

* Last Updated By
* Last Updated Date

When clicked, the link opens the attachment in a new browser tab or window depending on the browser settings.

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, these links opens the attachment in a new browser tab or window depending on the browser settings. 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 18–1 shows an example of an attachment field in a page or page segment with attachments.

Figure 18–1 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
  * Last Updated By
  * Last Updated Date

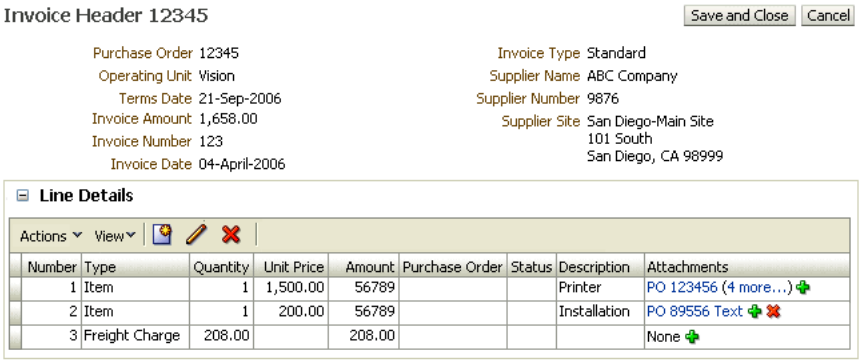
  When clicked, the link opens the attachment in a new browser tab or window depending on the browser settings.

  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, these links opens the attachment in a new browser tab or window depending on the browser settings. 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 18–2 shows an example of an attachment column in a table.

Figure 18–2 Attachment Column in a 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

This 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 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
- Type (required field that determines the type of Attachment)
  - File (default)
  - Text
  - URL
  - Repository File/Folder
- Category

  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.
- File Name or URL (required field)

  If the value is an existing attachment, a link is shown that opens the attachment when selected. The link opens in a new browser tab or window depending on the browser settings. 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

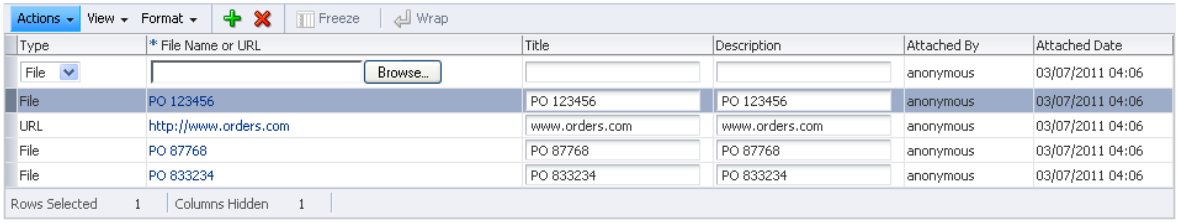
  The field to include additional information on the attachment.
- Last Updated By

  Shows the name of the user that last updated the attachment relationship.
- Last Updated Date

  Shows the date on which the attachment relationship was updated.

Figure 18–3 is an example of an Attachments table with the `repositoryMode` attribute set to *false*.

Figure 18–3 Attachments Table



18.2 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.

Before you begin:

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:

Example 18–1 Snippet for jazn-data.xml File

```
<grant>
   <grantee>
      <codesource>
       <url>file:${atgpf.oracle.home}/atgpf/modules/oracle.applcore.attachments_
       11.1.1/Attachments-Model.jar</url>
      </codesource>
   </grantee>
   <permissions>
     <permission>
       <class>oracle.security.jps.service.credstore
         .CredentialAccessPermission</class>
       <name>context=SYSTEM,mapName=oracle.wsm.security,
         keyName=keystore-csf-key</name>
       <actions>read</actions>
     </permission>
     <permission>
       <class>oracle.security.jps.service.credstore
         .CredentialAccessPermission</class>
       <name>context=SYSTEM,mapName=oracle.wsm.security,
             keyName=enc-csf-key</name>
         <actions>read</actions>
      </permission>
   </permissions>
</grant>
```

> **Note:** Be aware that the URLs and <class> entries cannot contain any spaces or line breaks.

3. Create a Content Repository connection in JDeveloper.

Notes: Attachments can support multiple Content servers.

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 Propgagtion for Authentication.

For information about how to create a Content Repository connection in JDeveloper, see Section 53.1, "Creating a Content Repository Connection".

For additional information see the "Integrating Content" chapter of the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

4. 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 in order 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. For more information, see "Internal Security: Security Groups, Roles, and Permissions " in *Oracle Universal Content Management: Managing Security and User Access, 10g Release 3 (10.1.3.3.1)*.

18.2.1 How to Set 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 Section 18.2.2, "How to Create Attachment View Links."

Once 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, or as an attachment column in an applications table. This design-time setup is performed in your ViewController project. For more information about setting up your ViewController project for attachments, see Section 18.2.6, "How to Create an Attachments Field or an Attachments Table," Section 18.2.8, "How to Create an Attachments Column in an Applications Table," and Section 18.3, "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 Section 18.2.7, "What Happens When You Implement Attachments" and Section 18.2.10, "What Happens at Runtime."

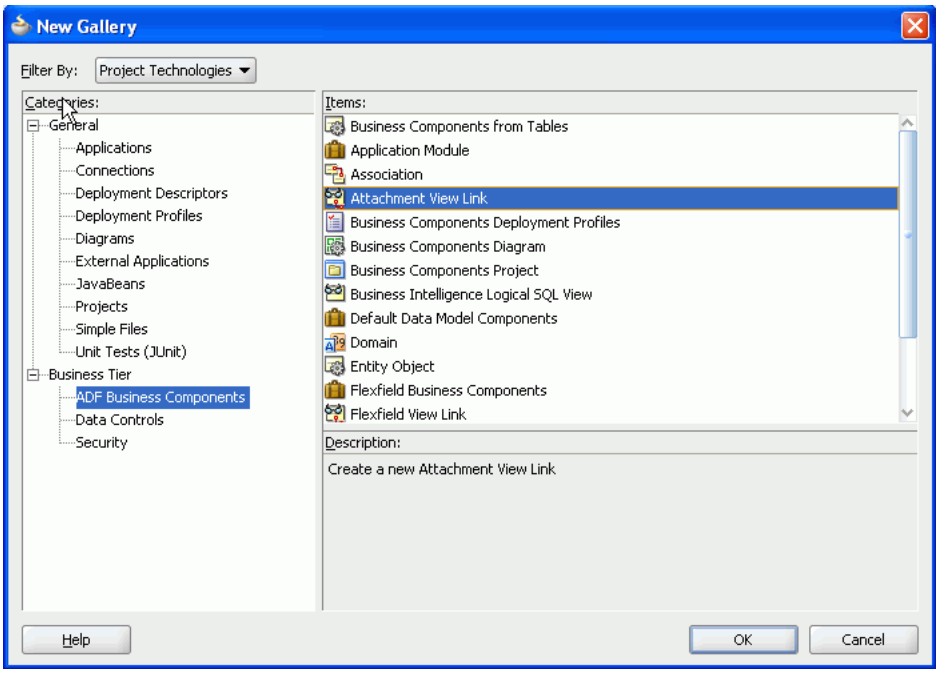
18.2.2 How to Create Attachment View Links

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:

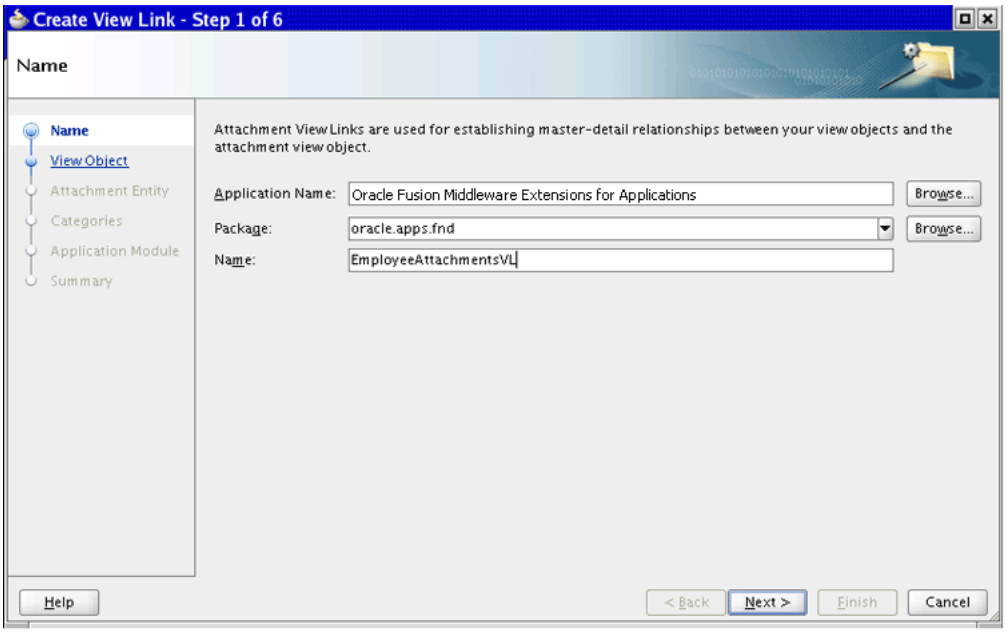
1. In JDeveloper, choose **Application Navigator > Model Project**. Right-click and choose **New** from the menu to open the New Gallery.
2. Choose the **Business Tier > ADF Business Components** category. Select the **Attachment View Link** item, as shown in Figure 18–4.

Figure 18–4 New Gallery Dialog



3. Click **OK** to access Step 1 of the Create Attachment View Link wizard, as shown in Figure 18–5.
4. Complete the following information:
   a. Select the name of the application that your view object belongs to.
   b. Select the package where the attachment view link will reside.
   c. Enter a unique name for the attachment view link.

Figure 18–5 Create Attachment View Link Wizard — Name (Step 1)

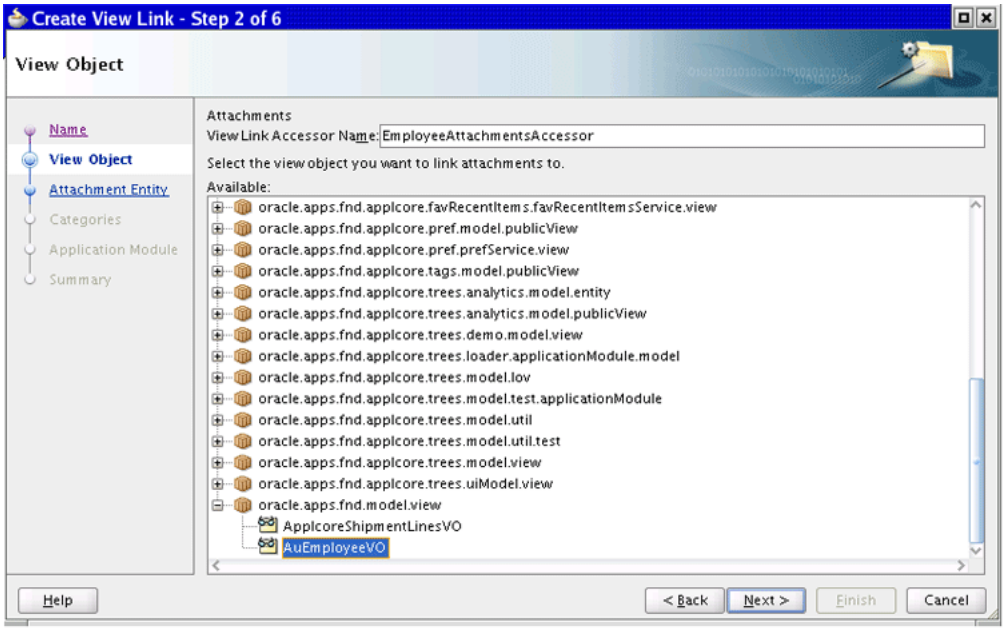


5. Click **Next** to access Step 2. The View Object dialog appears, as shown in Figure 18–6.

6. 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.

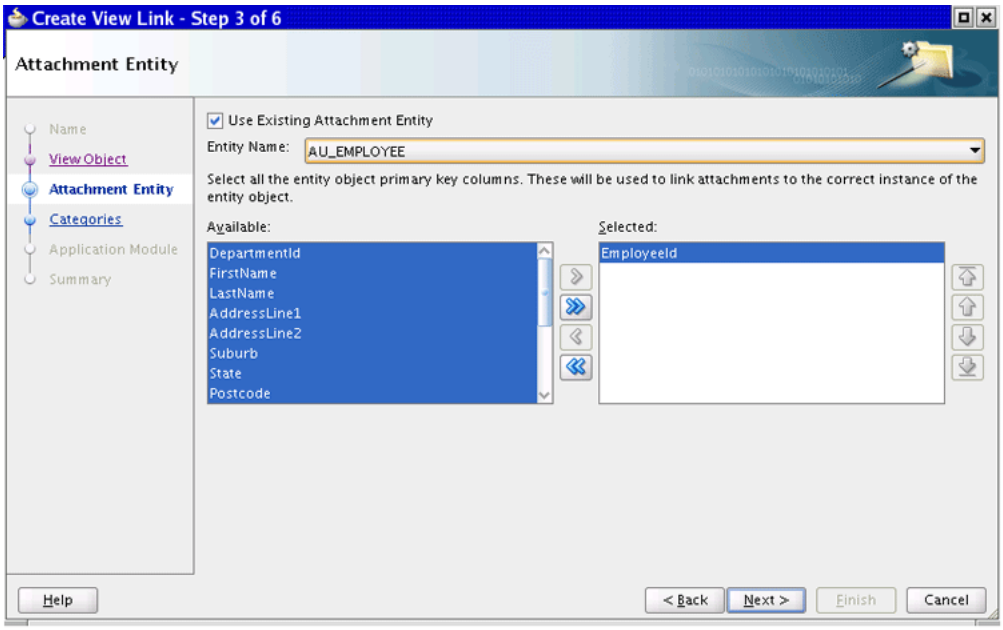
   Tip: There is no need for you to select a destination view object because the Attachment view link always uses the Attachments view object as the destination view object.

Figure 18–6 Create Attachment View Link Wizard — View Object (Step 2)



7. Click **Next** to access Step 3. The Attachment Entity dialog appears, as shown in Figure 18–7. The Attachment Entity is used to uniquely identify your business object.

Figure 18–7 Create Attachment View Link Wizard — Attachment Entity (Step 3)



8. Complete the following information:

   Use Existing Attachment Entity: When checked, the **Available / Selected** columns are automatically populated with the stored primary key values for the selected entity. When unchecked, the new entity is created in the `FND_DOCUMENT_ENTITIES` and `FND_DOCUMENT_ENTITIES_TL` tables.

   Entity Name: Enter a unique name for the entity you are adding attachments to.

   > **Note:** The entity name is used to map a business object to its attachments.

   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.

9. Click **Next** to access Step 4. The Categories dialog appears, as shown in Figure 18–8. (The **Display All Available Categories** checkbox is not selected when you first enter this page.)

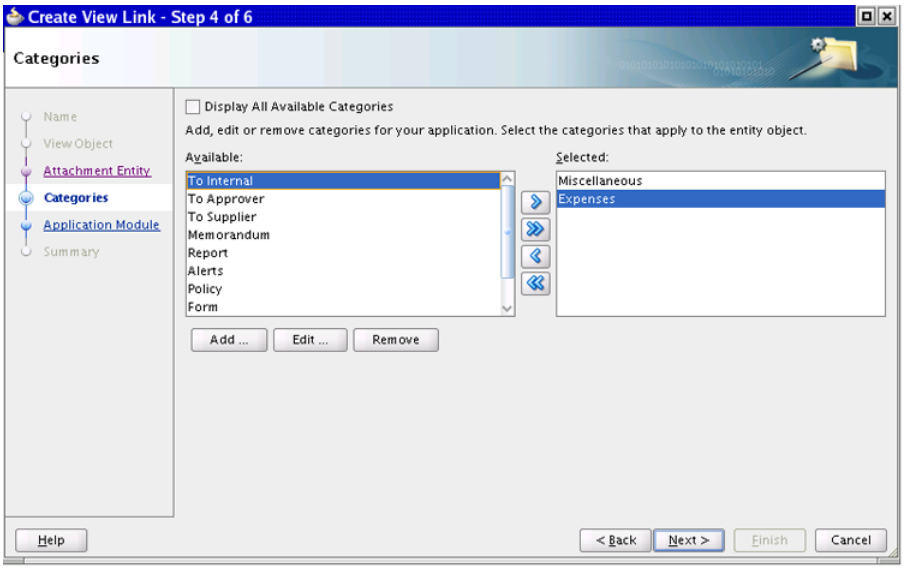
   Choose the categories that will be made available to the user in the Attachments runtime UI. The categories you select here must be assigned to your attachment entity. If they are not, they will not be visible in your UI.

   Categories that you create using this wizard are inserted into the `FND_DOCUMENT_CATEGORIES` and `FND_DOCUMENT_CATEGORIES_TL` tables against the Application that you selected in Step 1 of the Attachment View Link wizard. Categories that you edit are updated in the `FND_DOCUMENT_CATEGORIES` and `FND_DOCUMENT_CATEGORIES_TL` tables.

Notes:

- You can only add, edit or delete categories for the Application that you selected in Step 1 of the Attachment View Link wizard.
- The categories that you create here are not assigned automatically to their attachment entity. You must ensure that all categories you select in Step 9 are assigned to their Attachment Entity. For more information, see Section 18.2.5, "How to Assign Categories to the Attachment Entity."

Figure 18–8 Create Attachment View Link — Categories (Step 4)



Select the categories to be made available for this entity in the runtime UI.

Note: Use the **Add**, **Edit**, and **Delete** buttons to add, edit, or delete existing categories for the Application that you selected in Step 1. You are not able to edit or delete the seeded *Miscellaneous* FND category.

Tip: The end-user display name for the Category (`USER_NAME` from the `FND_DOCUMENT_CATEGORIES_TL` table) is shown in the shuttle.

Select **Display All Available Categories** to show all categories that are assigned to your attachment entity.

When you select **Display All Available Categories**, all other fields are automatically removed from the dialog. You will no longer see a list of categories and therefore, you will not be able to add, edit, or delete them. This selection, however, does not override the document category to entity mapping.

You should select this option if you want Categories, assigned to your attachment entity in the future, to appear in the Categories dropdown list at runtime. Selecting this option will ensure that customers can create and use their own Categories in your UI with minimal effort.

Deselect the **Display All Available Categories** checkbox to select a subset of categories that are used to further restrict the list of Attachments returned in the UI. Only the Attachments that are assigned to the selected categories will be retrieved in the UI. To select the required categories, highlight them in the **Available** column and shuttle them over to the **Selected** column.

Note: If you did not select the **Display All Available Categories** checkbox, you must select at least one category before proceeding to the final step in the wizard.

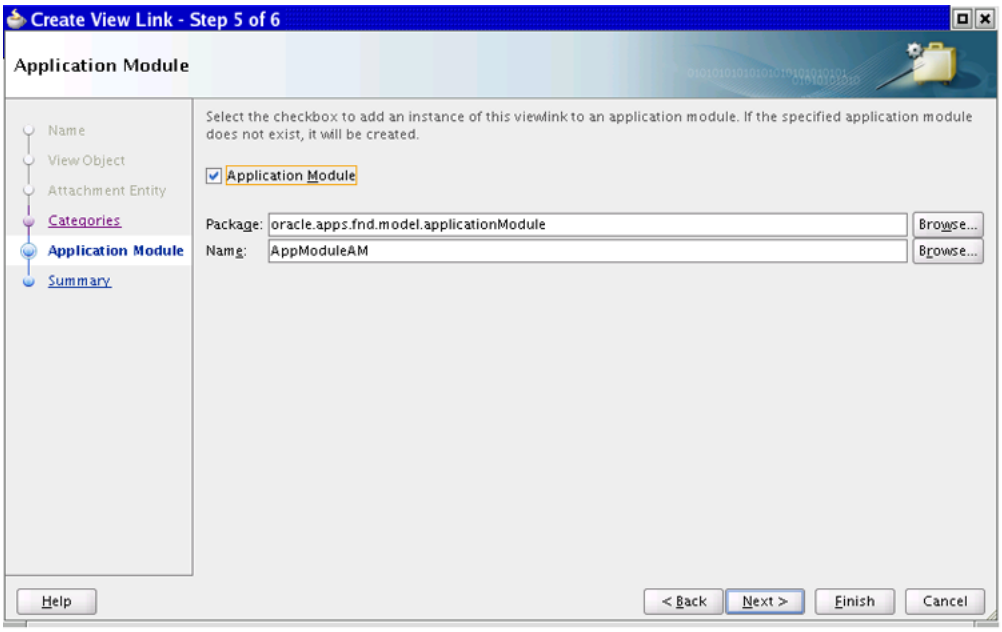
The selected list of categories will be stored as a custom property on the newly created attachment view link. The value will consist of a concatenated string of values. All the selected categories are concatenated in a comma-separated list.

Note: It is recommended that you choose the **Display All Available Categories** option to allow for customization of the Category list without code changes. Additionally, please 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 Section 18.7.1, "Attachment Category Data Security."

---

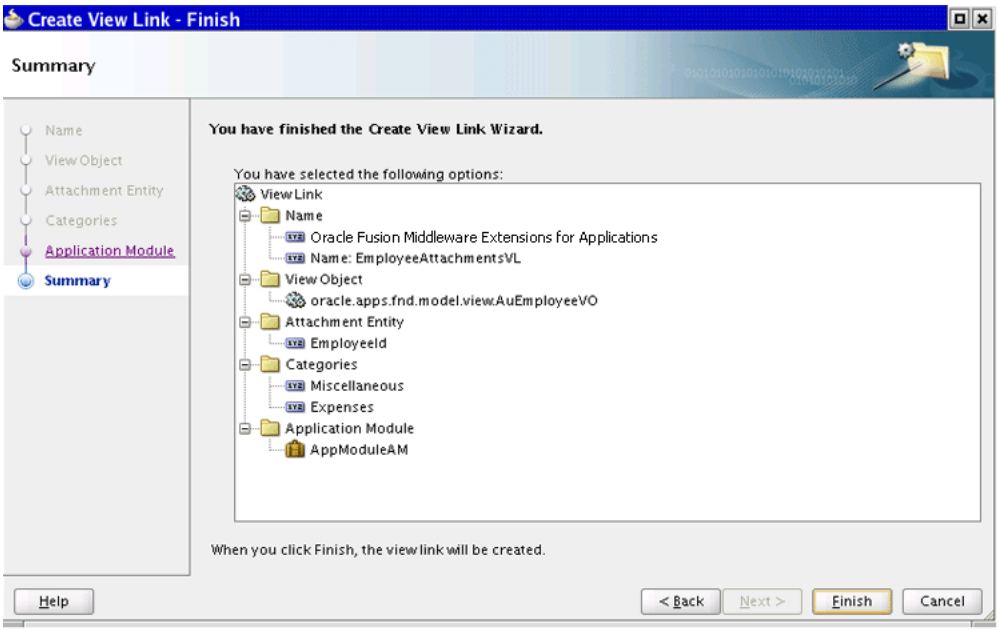
10. Click **Next** to access the Application Module page. The Application Module dialog appears, as shown in Figure 18–9. Select **Application Module** to create a new instance of the view object that you selected on the View Object page, and create an instance of the attachments view object as a child of the new instance.

Figure 18–9 Create Attachment View Link — Application Module (Step 5)



11. Click **Next** to access the Summary page, which is shown in Figure 18–10. Review your selections and click **Finish** to create your attachment view link.

Figure 18–10 Create Attachment View Link — Summary (Step 6)



12. 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. Save your changes.

18.2.3 What Happens When You Create an Attachment View Link

Once 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:

Example 18–2 Source WHERE Clause

```
(:Bind_InvoiceHeaderId = PK1_VALUE) AND
(:Bind_InvoiceLineId = PK2_VALUE) AND
(:Bind_AttachmentEntityName = ENTITY_NAME)
```

Example 18–3 Destination WHERE Clause

```
(:Bind_Pk1Value = PoInvoiceLines.INVOICE_HEADER_ID) AND
(:Bind_Pk2Value = PoInvoiceLines.INVOICE_LINE_ID) AND
(:Bind_EntityName = PO_INVOICES)
```

where `PO_INVOICES` is the value that you entered in the **Entity Name** field in Step 3 of the wizard.

18.2.4 How to Delete 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. Once you have access to this collection, you can loop through calling the `remove()` method of the collection, shown in Example 18–4.

Example 18–4 remove() Method

```
public void remove() {
      Row currRow = this.getCurrentRow();
      RowSet attachedItems = (RowSet) currRow.getAttribute("Attachments1");
      while (attachedItems.hasNext()) {
        Row currAttachment = attachedItems.next();
        currAttachment.remove();
      }
      super.remove();
 }
```

18.2.5 How to Assign 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 Section 18.6, "Integrating Attachments Task Flows into Oracle Fusion Functional Setup Manager." 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 55 for information about Attachments Seed Data Loaders.

18.2.6 How to Create an Attachments Field or an Attachments Table

This section describes how to create an Attachments field or table.

To create an Attachments field or an Attachments table:

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 18–11.

Figure 18–11 Create Attachments

Choose either **Attachment Field** or **Attachment Table**. 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.

   Based on the selection that you made in Step 3, either an **Attachment Field** or an **Attachment Table** is created and bound to the Detail Data Collection. The *mode* attribute is automatically set to **link** or **table** on the Attachment Component.

When using attachments in *link* mode, the Attachment component must be displayed inside a panelLabelAndMessage component and a partialTrigger needs to 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:

Example 18–5 Adding Event IDs to the PartialTriggers

```
<af:panelLabelAndMessage label="Attachment"
      partialTriggers="btnNext btnPrevious">
  <fnd:attachment mode="link" repositoryMode="true"
    attachmentModel="#{bindings.Attachments1}/>
</af:panelLabelAndMessage>
```

18.2.7 What Happens When You Implement Attachments

The following are examples of the generated source code:

Example 18–6 Source Code for an Attachment Field

```
<fnd:attachment mode="link"
   attachmentModel="#{bindings.AttachmentsIterator}"/>
```

Example 18–7 Source Code for an Attachment Table

```
<fnd:attachment mode="table"
   attachmentModel="#{bindings.AttachmentsIterator}"/>
```

Example 18–8 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.children}"/>
   columnLinkTableModel="#{bindings.AuEmployee1.collectionModel}"/>
</af:column>
```

Note: The bindings, as shown above, only show when the Attachments are bound to a data control.

18.2.8 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.

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.

> **Note:** The *mode* attribute is automatically set to **columnLink** on the Attachment component when creating an attachment column.

To create an attachments column when creating your applications table:

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:

   ```
   columnLinkTableModel="#{bindings.AuEmployee1.collectionModel}"
   ```

To create an attachments column in an existing applications table:

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}"
   ```

18.2.9 How to Set 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
- `RefreshCondition` for the task flow

To set 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">
```

To 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.

To 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.

18.2.10 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. Once 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`, `viewAllowed`, and `deleteAllowed` 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.

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.

18.3 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.

18.3.1 How to Configure the Attachments Component to Display Attachments for Multiple Entities

The following steps provide 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 Section 18.2.2, "How to Create Attachment View Links."

To configure your data model to display attachments for two or more business objects:

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.

   Close the view object.
5. Open the attachment view link.
6. 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 `Pk1Vlaue`) 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 18–9 and Example 18–10.

Example 18–9 Source Query

```
((:Bind_EmployeeId = PK1_VALUE) AND (:Bind_AttachmentEntityName = ENTITY_NAME))
OR
((:Bind_DeptNum = PK1_VALUE) AND (:Bind_AttachmentEntityName1 = ENTITY_NAME))
```

Example 18–10 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'))
```

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 18–11. (The added items are in bold).

Example 18–11 Source and Destination Attribute Arrays

```
<ViewLinkDefEnd
  Name="sourceEnd"
  Cardinality="1"
  Owner="oracle.apps.model.FndDemoDeptEmpVO"
  Source="true">
  <AttrArray Name="Attributes">
    <Item Value="oracle.apps.model.FndDemoDeptEmpVO.DeptNum"/>
    <Item Value="oracle.apps.model.FndDemoDeptEmpVO.AttachmentEntityName"/>
    <Item Value="oracle.apps.model.FndDemoDeptEmpVO.AttachmentEntityName1"/>
    <Item Value="oracle.apps.model.FndDemoDeptEmpVO.EmployeeId"/>
  </AttrArray>
</ViewLinkDefEnd>
<ViewLinkDefEnd
  Name="destEnd"
  Cardinality="-1"
  Owner="oracle.apps.fnd.applcore.attachments.uiModel.view.AttachmentsVO">
  <AttrArray Name="Attributes">
    <Item
Value="oracle.apps.fnd.applcore.attachments.uiModel.view.AttachmentsVO.Pk1Value"/>
    <Item
Value="oracle.apps.fnd.applcore.attachments.uiModel.view.AttachmentsVO.EntityName"
/>
    <Item
Value="oracle.apps.fnd.applcore.attachments.uiModel.view.AttachmentsVO.EntityName"
/>
    <Item
Value="oracle.apps.fnd.applcore.attachments.uiModel.view.AttachmentsVO.Pk1Value"/>
  </AttrArray>
```

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.

To configure your Attachments component to display attachments for two or more business objects:

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.

18.4 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. Refer to Table 18–1 for 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 or not a user is aware of the content repository. A more advanced UI is displayed to users who are aware of the content repository (`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 18–1 Attachment Component Properties

Property	Description	Default Value
Mode	String. Acceptable values are *link*, *columnLink*, and *table*.	The `mode` property is set automatically at the time the component is created.
RepositoryMode	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.	False
AttachFolderAllowed	Not used.	N/A
AttachLatestVersion	Not used.	N/A
ApprovalEnabled	Boolean: *true* / *false*. True: Adds the Status column to the Attachments table.	False
DeleteMessage	String. Message to be displayed on Delete Attachment confirmation page.	Are you sure you want to delete the selected attachment(s)?
Rows	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.	10
FileTypeEnabled	Boolean: *true* / *false*. True: Display attachment type of "File" in the Type choice list. False: the component is hidden.	True
TextTypeEnabled	Boolean: *true* / *false*. True: Display attachment type of "Text" in the Type choice list.	True
UrlTypeEnabled	Boolean: *true* / *false*. True: Display attachment type of "URL" in the Type choice list.	True

Table 18–1 (Cont.) Attachment Component Properties

Property	Description	Default Value
NumAttachmentsDisplayed	Number. Maximum number of attachments displayed as links in the hover popup when mode is set to link or columnLink.	3
AddAllowed	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.	True
AddEnabled	Boolean: *true / false*. True: shows the icon in the toolbar. False: hides the icon in the toolbar.	True
ActionEntity	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.	
UpdateAllowed	Boolean: *true / false*. Enables/disables the update icons in the attachments table toolbar. These include the "Check In", "Check Out" and "Cancel Check Out" 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.	True
UpdateEnabled	Boolean: *true / false*. Controls show/hide of the update icons in the attachments table toolbar. These include the"Check In", "Check Out" and "Cancel Check Out" icons. True: shows the update icons in the toolbar. False: hides the update icons in the toolbar.	True
InsertMultiple	Boolean: *true / false*. Determines whether more than one attachment can be assigned. False: User can only add a one attachment.	True

Table 18–1 (Cont.) Attachment Component Properties

Property	Description	Default Value
DeleteAllowed	Boolean: *true* / *false*. Enables/disables the "Delete" 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.	True
DeleteEnabled	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.	True
ViewAllowed	Boolean: *true* / *false*. True: Viewing attachments is allowed. False: Links for viewing documents in Attachments tables are disabled.	True
DefaultCategory	String. Value selected as the default in the Category poplist when adding new attachments.	
Rendered	Boolean: *true* / *false*. True: Component is rendered in the page. False: the component is not rendered.	True
Id	String. The identifier of the component.	Auto-generated
Visible	Boolean: *true* / *false*. True: the component is displayed in the page. False: the component is hidden.	True
ShortDesc	String. The short description of the component. This text is commonly used by user agents to display tooltip help text.	
Label	String. Specifies the title for the Attachment popup.	Attachment
AutoHeightRows	Number. Sets the maximum number of rows that the table height will automatically adjust to depending on the amount of data in the table.	10
ContentDelivery	String. This property is only used in conjunction with the AutoHeightRows property. If AutoHeightRows is not set, this property should be not be set either.	Immediate
RowBandingInterval	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.	

Table 18–1 (Cont.) Attachment Component Properties

Property	Description	Default Value
ColumnBandingInterval	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.	
ShowCategory	Boolean: *true / false*. Controls show/hide of the Category column in the Attachments table.	True
UpdateCategoryList	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.	
SecondaryToolbarRendered	Boolean: *true / false*. True: shows the secondary toolbar. False: hides the icon in the toolbar.	False

18.5 Setting Up Miscellaneous Attachments Features

When the `repositoryMode` property is set to *true*, the following features become available:

- Custom Actions
- Approvals

18.5.1 Custom Actions

Four facets (placeholders) are provided on the Attachment 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.

18.5.2 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):

Lookup Code	Meaning
APPROVED	Approved
REJECTED	Rejected
REVIEWED	Reviewed
SUBMITTED_FOR_APPROVAL	Submitted for Approval
SUBMITTED_FOR_REVIEW	Submitted for Review
UNAPPROVED	Unapproved

Note: The **Status** column is a read-only column when displayed in the Attachments Table.

18.6 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 Chapter 49, "Implementing Function Security."

Table 18–2 lists the task flows and their parameters.

Table 18–2 Attachments Task Flows and Parameters

Task Flow Name	Task Flow XML	Parameters Passed	Behavior	Comments
Manage Attachment Entities	/WEB-INF/oracle/apps/fnd/applcore/attachments/publicUi/flow/ManageAttachmentEntities.xml#ManageAttachmentEntities	[moduleType] [moduleKey] [pageTitle]	Search and edit Attachment entities.	NA
Manage Attachment Categories	/WEB-INF/oracle/apps/fnd/applcore/attachments/publicUi/flow/ManageAttachmentCategories.xml#ManageAttachmentCategories	[moduleType] [moduleKey] [pageTitle]	Search and edit Attachment categories.	NA

For more information about task flows, see *Oracle Fusion Applications Common Implementation Guide*.

18.7 Securing Attachments

All attachments users must be assigned the "AttachmentsUser" role in order 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 following two types of security:

- Attachment Category data security

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 Section 18.7.2, "File Sharing."

18.7.1 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 Section 18.2.5, "How to Assign 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 18–3.

Table 18–3 Actions Seeded for Attachments

Action	FUNCTION_NAME	USER_FUNCTION_NAME
Seeded Read Action	FND_READ_APPLICATION_ATTACHMENT_DATA	Read Application Attachment
Seeded Update Action	FND_UPDATE_APPLICATION_ATTACHMENT_DATA	Update Application Attachment
Seeded Delete Action	FND_DELETE_APPLICATION_ATTACHMENT_DATA	Delete Application Attachment

18.7.1.1 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 18–3) to a role, but a role must be assigned the read action in order 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 once 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 Chapter 48, "Implementing Oracle Fusion Data Security."

18.7.2 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 Section 18.8.1, "How to Use 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.

18.7.3 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.

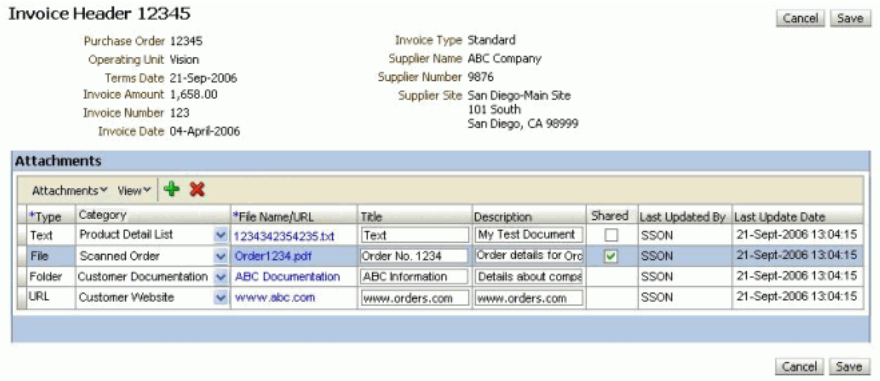
18.8 Using Attachments (Runtime)

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.

18.8.1 How to Use 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 18–12.

Figure 18–12 Attachment Table with Shared Column



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 18–12. 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*.

18.8.2 How to Update 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.

Icon	Function	File	Property
	Check Out	versioncheckout_ena.png	repositoryMode = true
	Check In	versioncheckin_ena.png	repositoryMode = true
	Cancel Check Out	versionrollback_ena.png	repositoryMode = true

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.

18.8.2.1 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*:

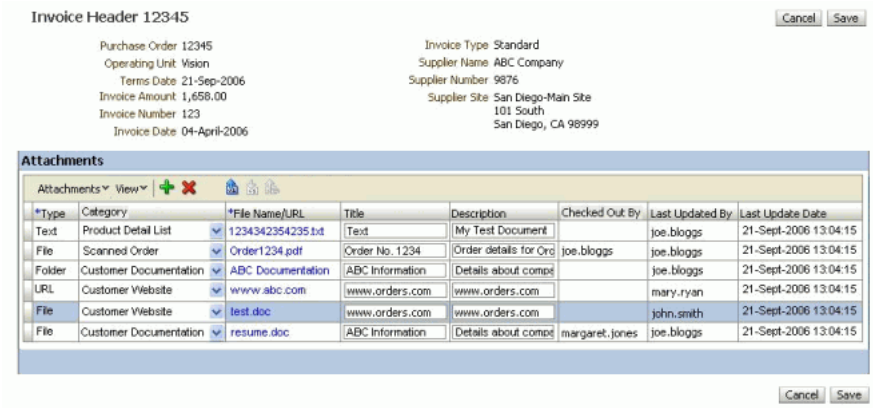
- The *Category*, *Title*, and *Description* columns in the Attachments table are updateable.

  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 18–13.

These buttons are enabled or disabled based on the type of attachment and the checked out status of File and Text type attachments.

Figure 18–13 Attachment Table with Update Functions set to True



- If `repositoryMode` = *false*:
  - The *Check In*, *Check Out*, and *Cancel Check Out* buttons are not rendered in the Attachments table toolbar.

Figure 18–14 Attachment Table with Update Functions set to False

`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.

18.8.2.2 Determining 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.

18.8.2.3 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 `repostoryMode` = *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.

18.8.3 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:

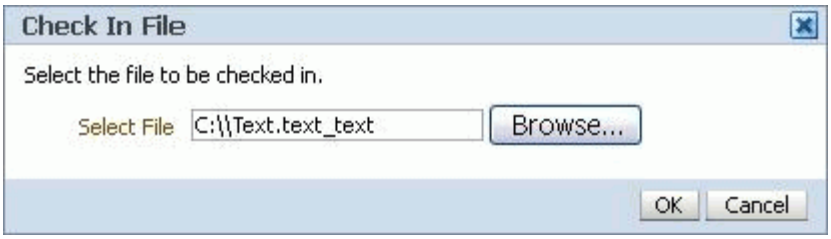
1. Highlight the File attachment in the Attachments table. Click the **Check Out** button located on the toolbar.

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 18–15.

Figure 18–15 Check In File Dialog



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.

19

Organizing Hierarchical Data with Tree Structures

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:

19.1 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 Oracle Fusion 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.

19.1.1 Understanding Tree Structures, Trees, and Tree Versions

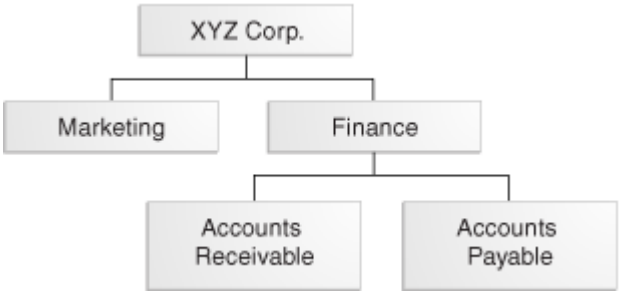
A *tree structure* is a way of describing a hierarchy. A *tree* is an instance of this hierarchy. Every tree structure contains a tree. Trees may have one or more versions. Each *tree version* contains at least one *root node*; that is, a member that has no superior. (Occasionally, a tree version may 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 Figure 19–1, 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.

Figure 19–1 Example of a Tree



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 for 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 by way of its nodes. Multiple data sources can be associated with a tree structure and can have well-defined relationships among them. Using the example in Figure 19–1, 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.

Table 19–1 lists other commonly used tree terms and their descriptions.

Table 19–1 Common Tree Terminology

Term	Description
Depth	The depth of a node is the length of the path from the root to the node. The root node is at depth zero.
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 have a storage home, only customers should store the node in an entity.
Tree node	A node that is included in a tree.

Table 19–1 (Cont.) Common Tree Terminology

Term	Description
Tree node type	A tree node has a node type. Node types can be any one of the following: ■ Single - Indicates that the node is a value by itself. For example, a tree node for Employee "Larry Ellison" or Employee "Steve Jobs" in an employee hierarchy. ■ Range - Indicates that the node represents a range of values and possibly could have many children. For example, a tree node representing account numbers 10000 to 99999. ■ Referenced Tree - Indicates that the tree node is actually another tree whose nodes are not physically stored in this tree. For example, a geographic hierarchy for the United States can be referenced in a World geographic hierarchy.
Tree levels	Provide a way to organize tree nodes. In most trees, all nodes at the same level represent the same kind of information. For example, in a tree that reflects the organizational hierarchy, all division nodes appear on one level and all department nodes on another. Similarly, in a tree that organizes a user's product catalog, the nodes representing individual products might appear on one level and the nodes representing product lines on the next higher level. When levels are not used, the nodes in the tree have no real hierarchy or reporting structure but do form a logical summarization structure. Strictly enforced levels mean that the named levels describe each node's position in the tree. This is natural for most hierarchies. Loosely enforced levels mean that the nodes at the same visual level of indentation do not all represent the same kind of information, or nodes representing the same kind of information appear at multiple levels. With loosely enforced levels, users assign a level to each node individually; the level is not tied to a particular visual position.
Tree structure access	The set of rules that control access to a tree structure.
Tree access	The set of rules that control access to a tree.
Tree node access	The set of rules that control access to a particular node (and its subtree) within a given tree version.
Effective dates	Enable users to specify new objects, departments, reporting relationships, or organizational structures in advance and have them take effect automatically. Users also can use trees with past, present, or future effective dates when reporting on current or historic data.
Reference data set determinant (external)	A value that determines which reference data set will be used for each reference data object. Business units, regulatory regions, and reference data sets all can determine which reference data sets are valid for the creation of a transaction or reference data object.
Audit	A process that runs a series of tests against tree metadata and tree data to validate its integrity.

19.2 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."

To create the launch page:

1. Configure a UIShell launcher page for your **ViewController** project using the procedure described in Chapter 14.2, "Populating a UI Shell."
2. Add a taskflow entry in the ADF menu as a node with 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.

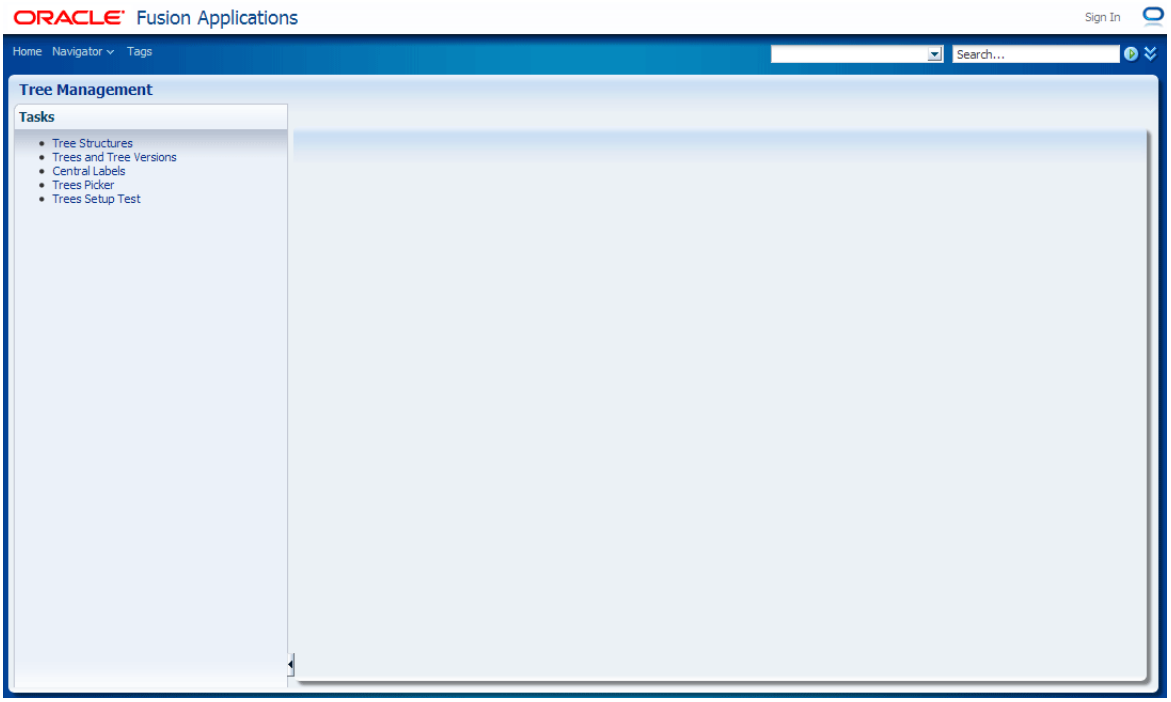
   - **label** - `#{applcoreBundle.TASKS}`
   - **Task Type** - `defaultRegional`

- **taskFlowId** - `/WEB-INF/oracle/apps/fnd/applcore/patterns/uishell/ui/publicFlow/TasksList.xml#TasksList`
- **Disclosed** - `true`

6. At the Set Run Configuration window, click **OK**.

   The trees application launch page opens in a browser window, as shown in Figure 19–2.

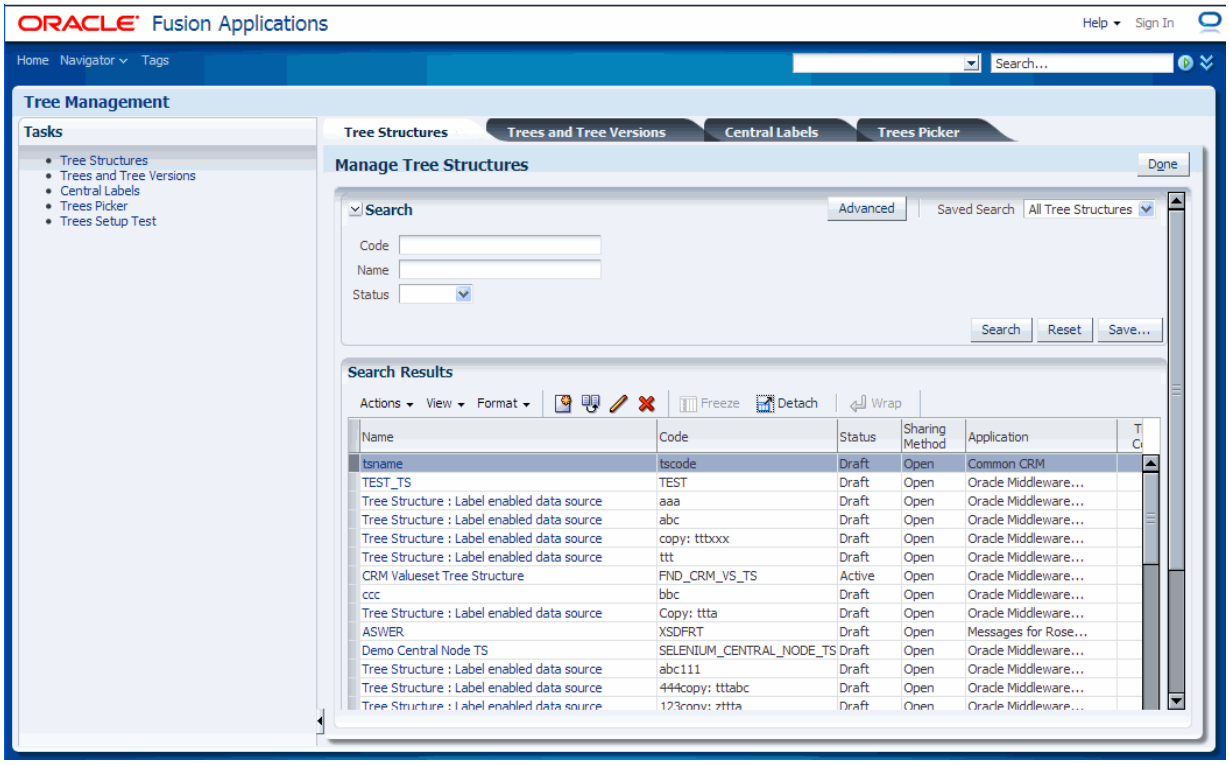
Figure 19–2 Trees Application Launch Page



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.

   Figure 19–3 shows the launch page with all applications open, and the Tree Structure application's summary page displayed.

Figure 19–3 Trees Application Launch Page with Applications Open



19.3 Working with Tree Structures

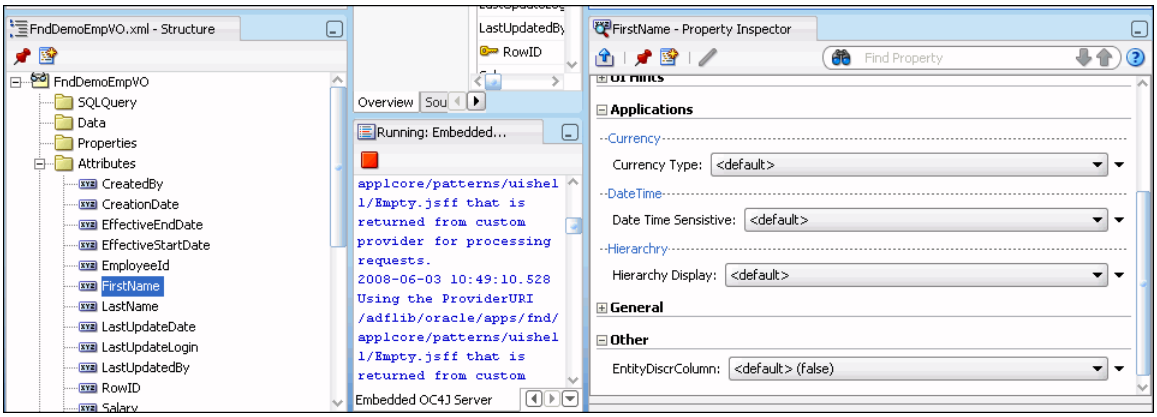
Working with tree structures includes the following tasks:

- 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

19.3.1 How to Manage 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`, as shown in Figure 19–4.

Figure 19–4 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`.

19.3.2 How to Specify 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 you 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 Table 19–2.

Table 19–2 Parameter Syntax

Attribute	Syntax
Tree Structure	#{treeStructure.ATTR_NAME_WITH FIRSTCHAR_IN_LOWER CASE} For example, `#{treeStructure.treeStructureCode}`.
Tree	#{tree.ATTR_NAME_WITH FIRSTCHAR_IN_LOWER CASE} For example, `#{tree.treeCode}`.
Tree Version	#{treeVersion.ATTR_NAME_WITH FIRSTCHAR_IN_LOWER CASE} For example, `#{treeVersion.treeVersionId}`.

Notes: 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.

19.3.2.1 Implementing Use Cases

This section includes an example use case and discusses basic use cases and their settings.

19.3.2.1.1 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.

19.3.2.1.2 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 19–5 View Object Setup Wizard

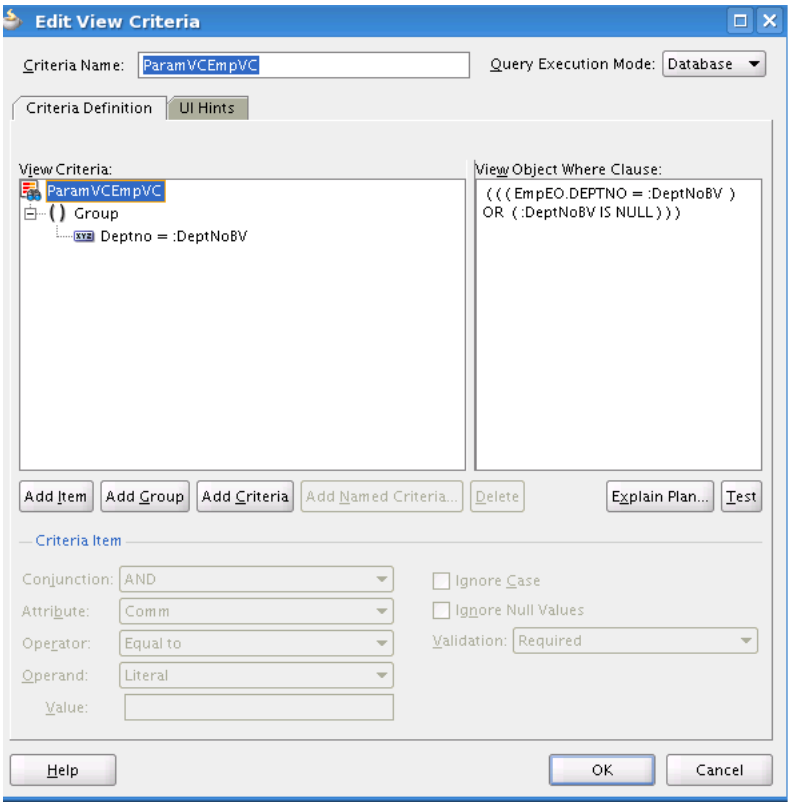
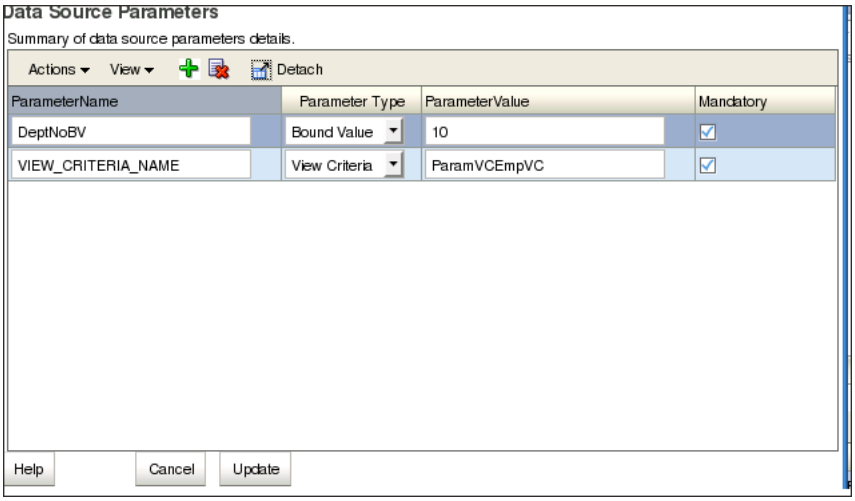


Figure 19–6 Parameters in Data Source Parameter UI



Data source has a WHERE clause using a bind variable:

Figure 19–7 View Object Setup Wizard

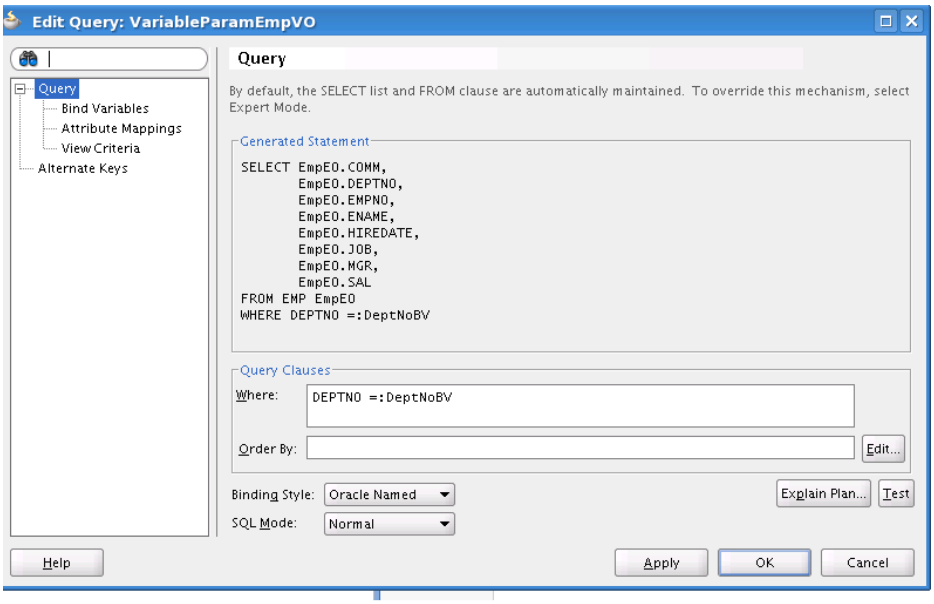
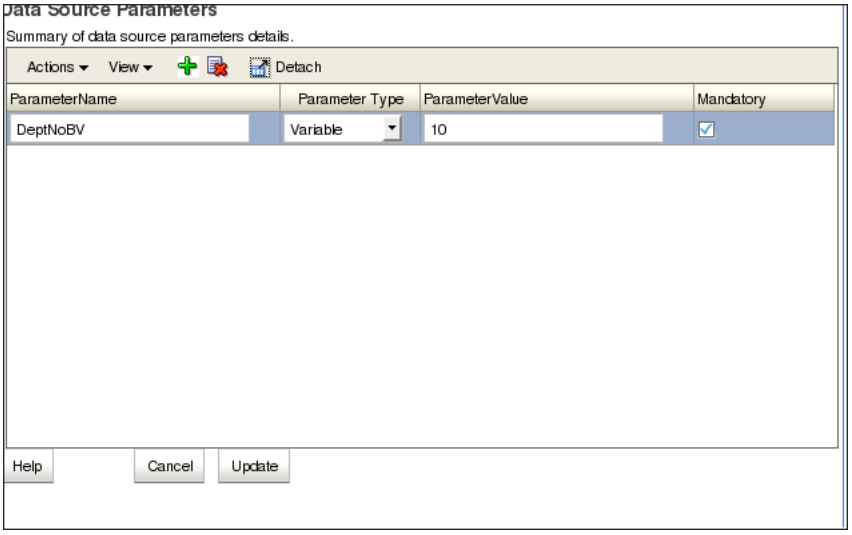


Figure 19–8 Parameters in Data Source Parameter UI



Data source has a view criteria defined with a bind variable for special parameters:

Figure 19–9 View Object Setup Wizard (View Criteria)

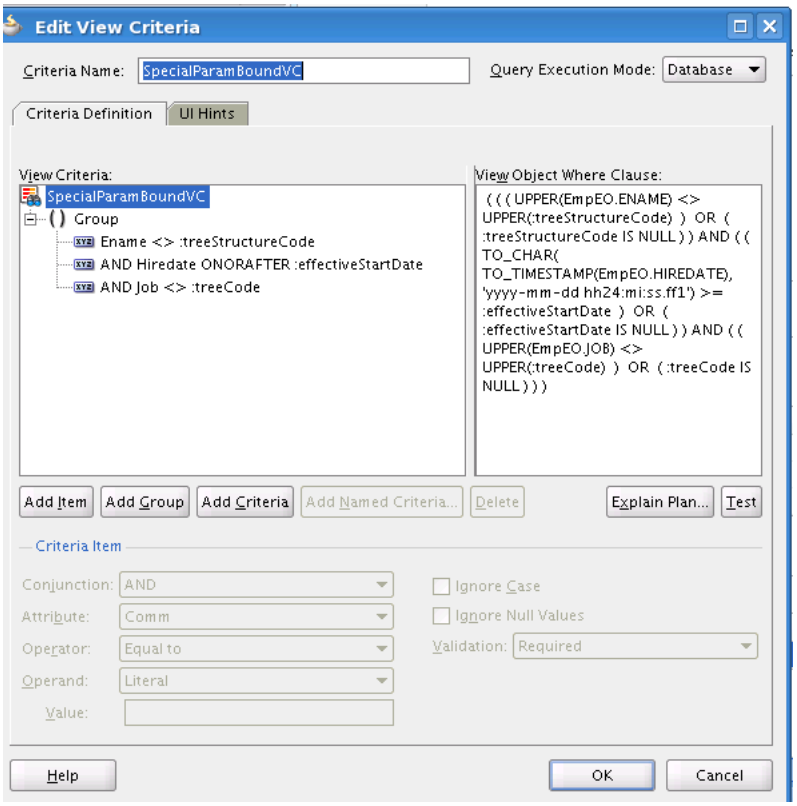
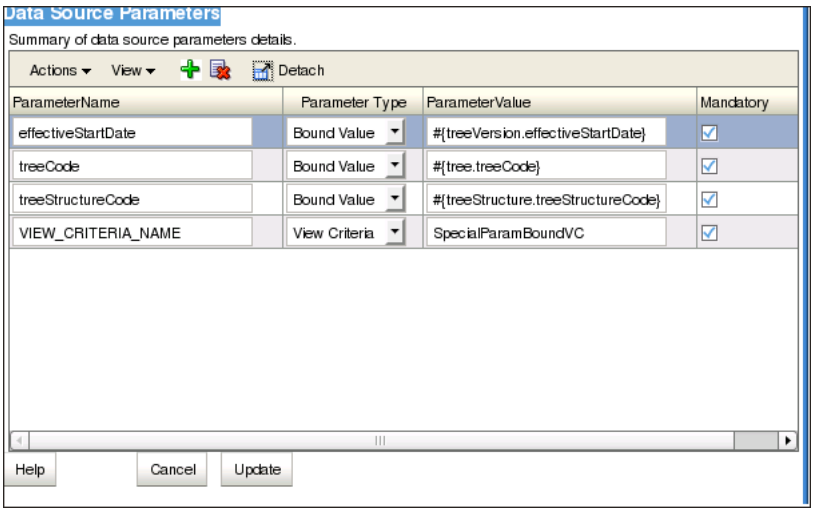


Figure 19–10 Parameters in Data Source Parameter UI



Data source has a WHERE clause using a bind variable for special parameters:

Figure 19–11 View Object Setup Wizard (WHERE Clause)

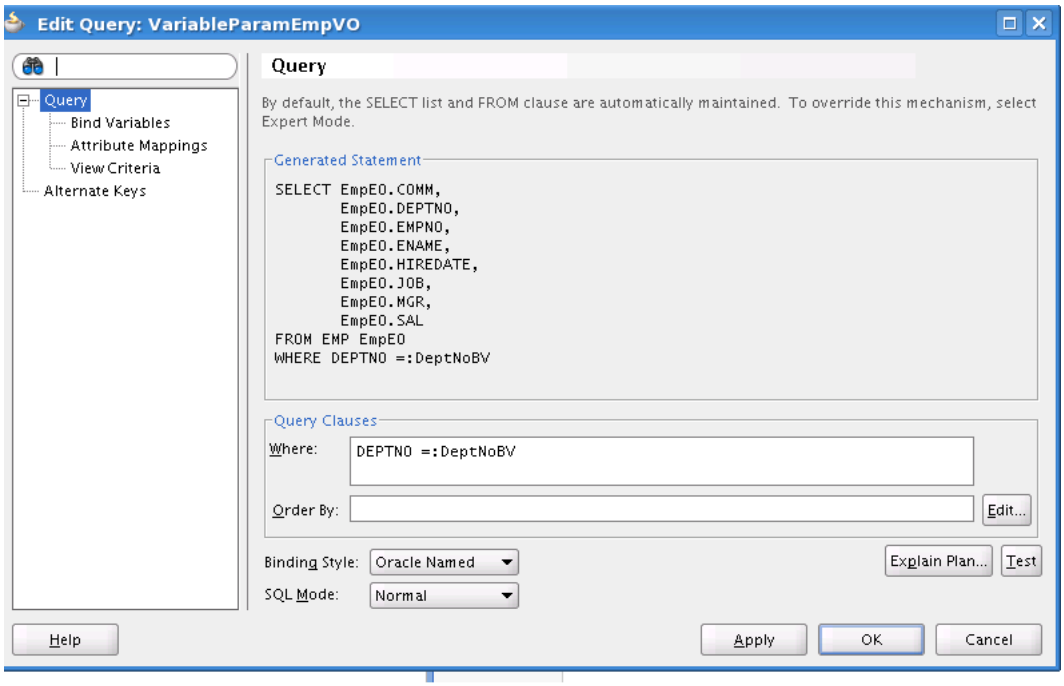
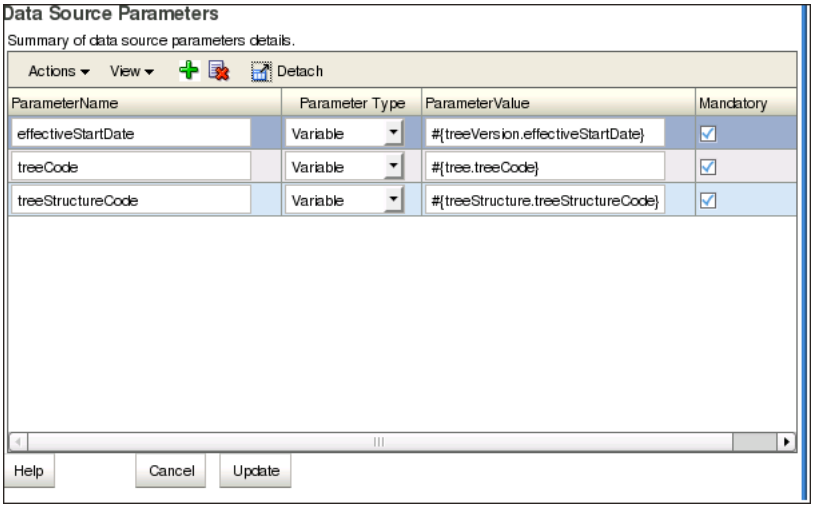


Figure 19–12 Parameters in Data Source Parameter UI



19.3.3 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:

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**.

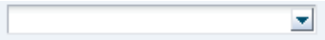
   All tree structures matching your search criteria appear in the Results area of the page.

Clicking **Advanced** enables you to perform an advanced search by specifying additional options, such as adding fields to search. You also can save your search criteria for future use.

19.3.4 How to Use the Search Field

Throughout the trees application you will see a search field located to the right of many field names, as shown in Figure 19–13.

Figure 19–13 Search Field



Clicking the down arrow displays a search field dropdown list that contains the available values for that field. You can select from the list, or search for other values. For example, Figure 19–14 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.

Figure 19–14 Search Field Dropdown List

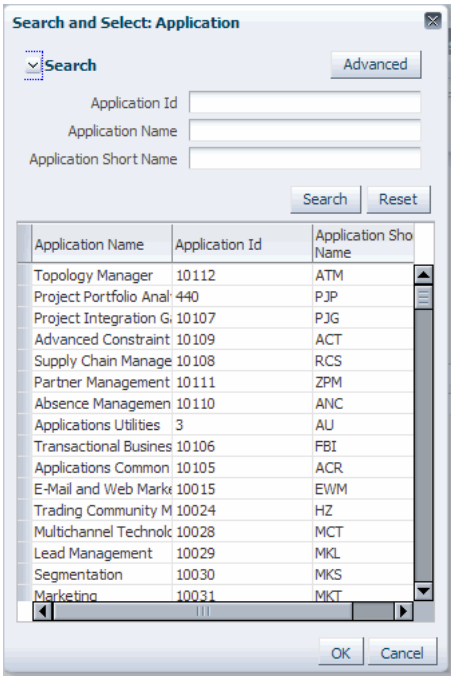
From each search field dropdown list, you can do one of the following:

- Select a value from the list that displays.
- Click the **Search** link to search for a value not in the list.

If you select a value from the list, the dropdown 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 Figure 19–15 opens:

Figure 19–15 Search-and-Select Window



You now can search for a value and then click **OK** to select it.

19.3.5 How to Create a Tree Structure

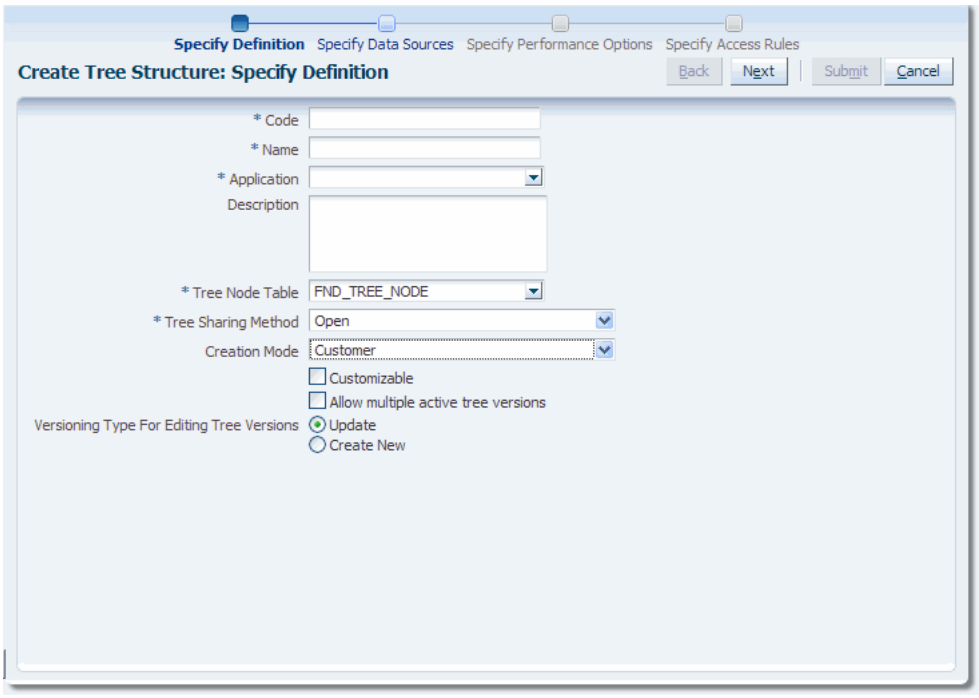
The following procedure explains how to create a new tree structure. The procedure assumes that the Tree Structure summary page is open in your web browser.

To create a tree structure:

1. Click the Create icon, or choose **Create** from the **Actions** dropdown menu.

   The Create Tree Structure: Specify Definition page, shown in Figure 19–16, opens.

Figure 19–16 Create Tree Structure: Specify Definition Page



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.

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.

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.

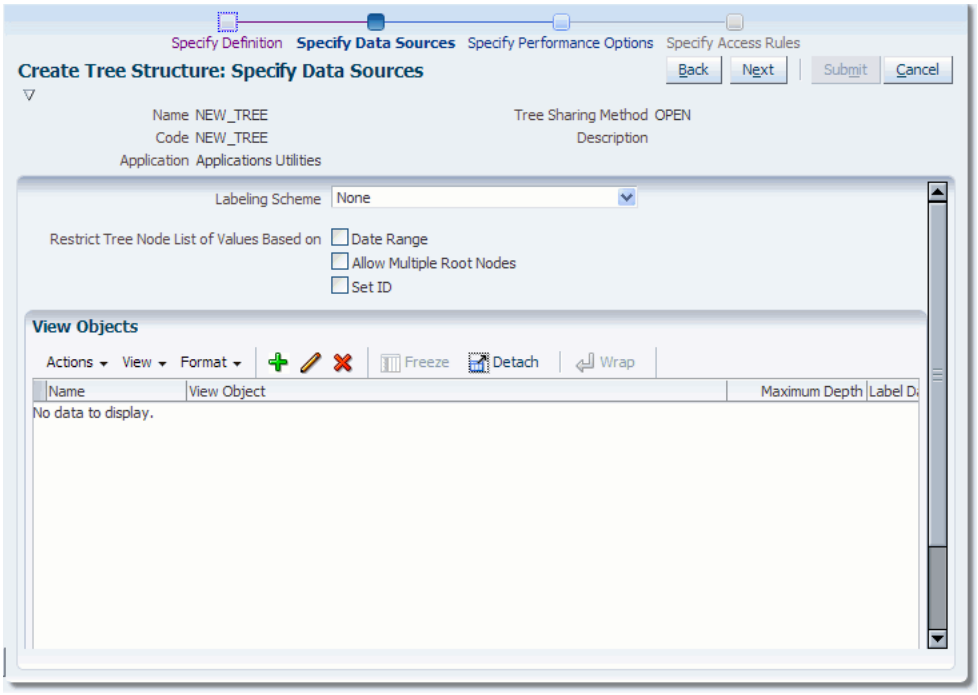
10. Select **Allow Multiple Active Tree Versions** to allow two or more tree versions to be in an ACTIVE state for the same date range.

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 Figure 19–17, opens.

Figure 19–17 Create Tree Structure: Specify Data Sources Page

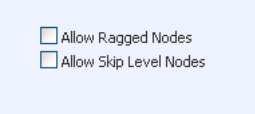


13. Select a labeling scheme:

    - **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 Figure 19–18.

Figure 19–18 Additional Labeling Scheme Options

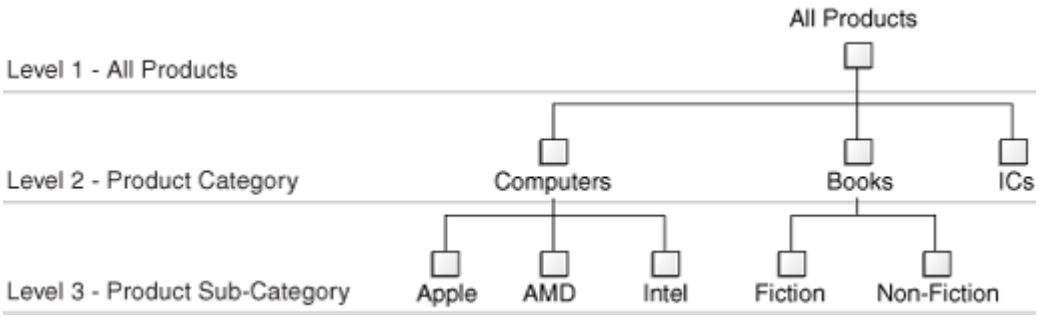


14. Select any or all of the following:

    - **Allow Multiple Root Nodes** - Allows you to add multiple root nodes when creating a tree version. (Not yet implemented.)
    - **Date Range** - Specifies whether a selection of nodes should be restricted to the same date range as the 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.)

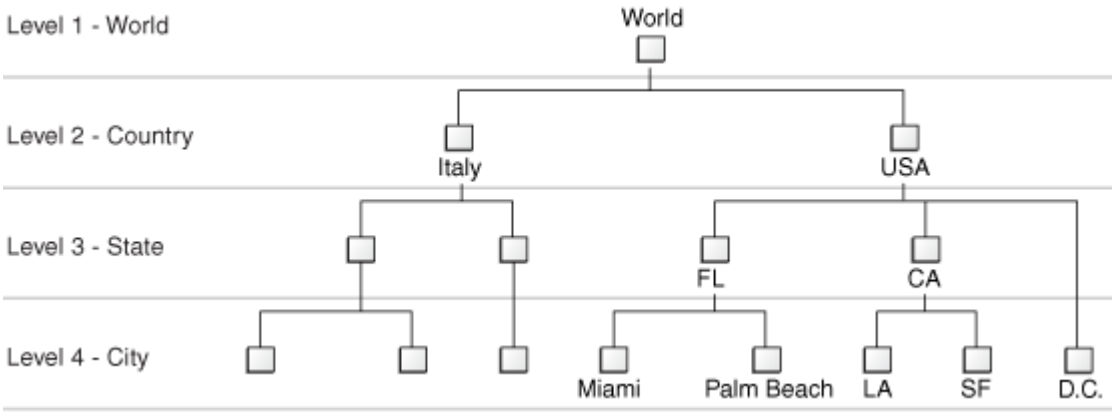
- **Allow Ragged Nodes** (Level, Depth, or Group only) - Specifies whether a hierarchy can be unbalanced; that is, if it can contain nodes that are not leaf nodes and contain no children. In Figure 19–19, "ICs" does not have any children, making its rooted path shorter than all the others in the hierarchy.

Figure 19–19 Example of a Ragged Hierarchy



- **Allow Skip Level Nodes** (Level, Depth, or Group only) - Specifies whether a hierarchy can have two nodes at the same level with parent nodes at different levels. In Figure 19–20, Washington, DC, does not have a node at the State level.

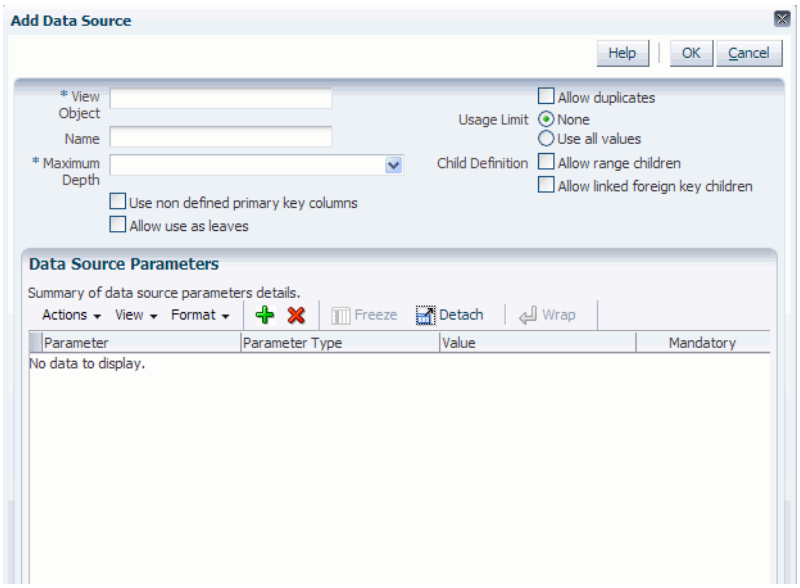
Figure 19–20 Example of a Skip-Level Hierarchy



15. Click the **Add** icon.

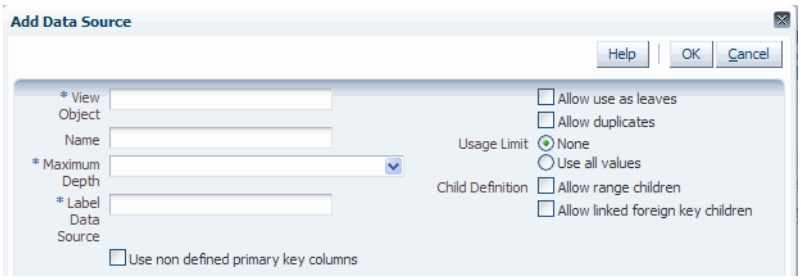
    The Add Data Source window, shown in Figure 19–21, opens.

Figure 19–21 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 Figure 19–22.

Figure 19–22 Additional Add Data Source Field



16. Enter the view object path.

    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 Section 19.6, "Managing Labels in the Generic Label Data Source."

17. Optionally, enter a name.

18. 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.

19. Enter the label data source path (Level, Depth, or Group only).

20. 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.

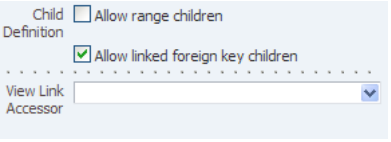
If selected, the additional fields shown in Figure 19–23 display.

Figure 19–23 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.
- 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.
- **Define Children By: Range** - indicates that the hierarchy supports nodes that are a range of values.
- **Define Children By: Foreign Key Relationship** - 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 Figure 19–24.

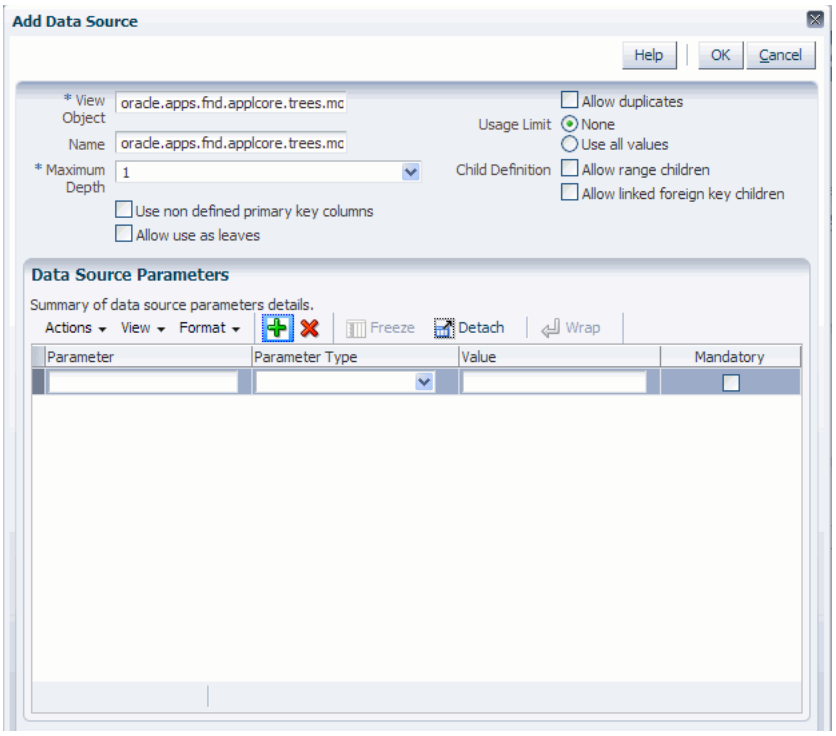
Figure 19–24 View Link Accessor Dropdown List



21. Click **Add** under **Data Source Parameters**.

    The window now displays data source parameters text-entry fields, as shown in Figure 19–25.

Figure 19–25 Data Source Parameters Text-Entry Fields



22. Enter a parameter, select a parameter type, and enter a value.

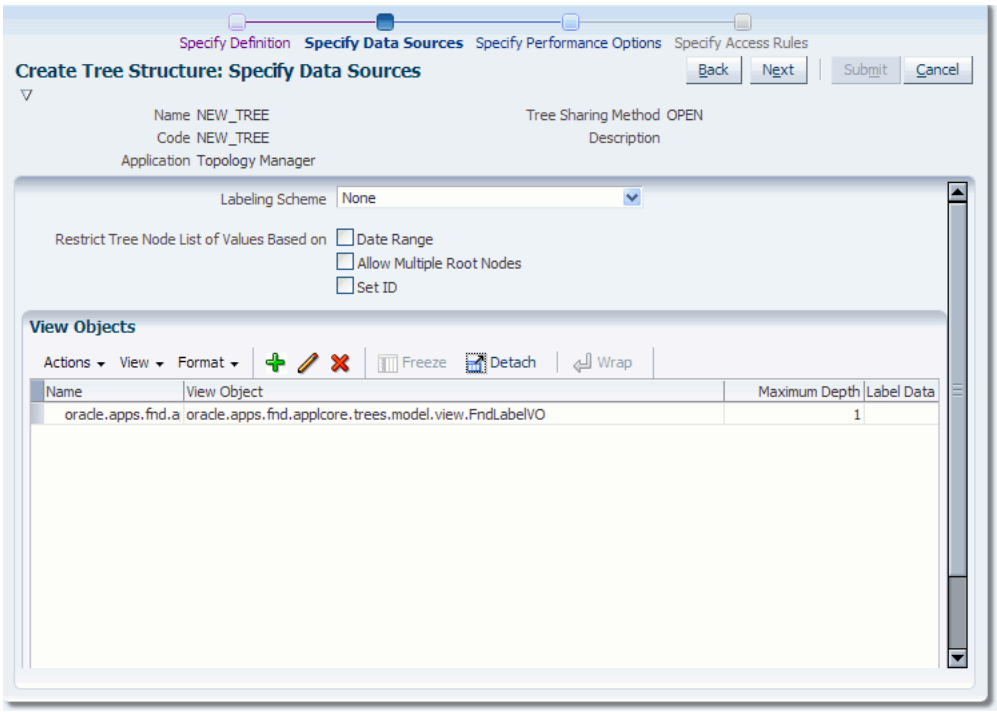
    When specified, a parameter applies to every version under that tree. Parameter values also can be overridden at the tree level. For more information, see Section 19.3.2, "How to Specify Data Source Parameters."

23. Select **Mandatory** if the parameter is to be required.

24. Click **OK**.

    The Create Tree Structure: Specify Data Sources page refreshes, displaying the view object, as shown in Figure 19–26.

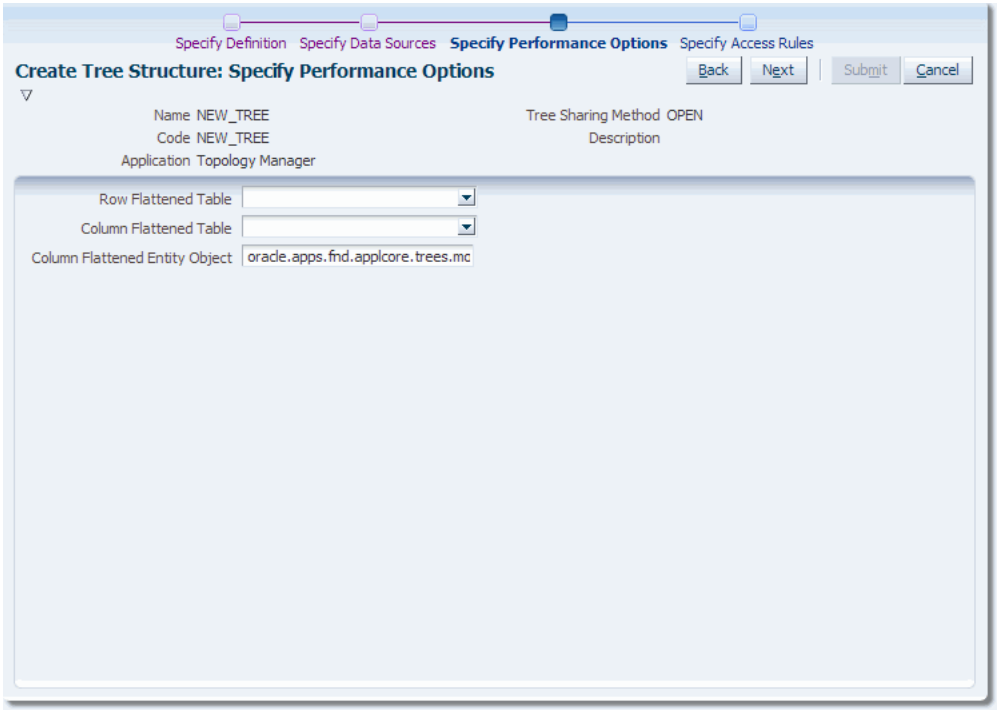
Figure 19–26 Create Tree Structure: Specify Data Sources Page with View Object



25. Click **Next**.

The Create Tree Structure: Specify Performance Options page opens, as shown in Figure 19–27.

Figure 19–27 Create Tree Structure: Specify Performance Options Page



26. Enter the name of an appropriate row-flattened table or click the down arrow to select or search for one.
27. Enter the name of an appropriate column-flattened table or click the down arrow to select or search for one.
28. Enter the name of an appropriate column-flattened entity object if the field does not already contain one.
29. Click **Next**.

    The Create Tree Structure: Specify Access Rules page opens.

    Note: The Create Tree Structure: Specify Access Rules page is not yet implemented.

30. Click **Submit**.
31. Click **OK** to close the Confirmation window.

19.3.6 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:

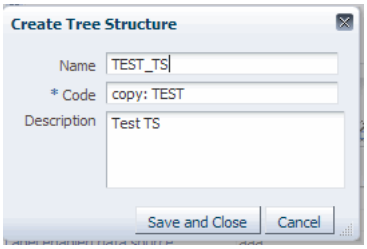
1. Select the tree structure you want to duplicate.

   See Section 19.3.3, "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 Figure 19–28:

Figure 19–28 Create Tree Structure Window



3. Enter a new name for the duplicate tree structure if you want to replace the that already displays in the field.
4. Enter a duplicate tree structure code if you want to replace the that already displays in the field.
5. Click **Save and Close** to create the duplicate.

19.3.7 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:

1. Select the tree structure you want to edit.

   See Section 19.3.3, "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.

19.3.8 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:

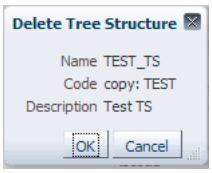
1. Select the tree structure you want to delete.

   See Section 19.3.3, "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 Figure 19–29:

Figure 19–29 Delete Tree Structure Warning Window



3. Do one of the following:
   - Click **Cancel** to cancel the operation.
   - Click **OK** to delete the tree structure.

19.3.9 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 Section 19.3.10, "How to Audit a Tree Structure," for more information about auditing.

To set the status of a tree structure:

1. Select a tree structure.

   See Section 19.3.3, "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.
4. Click **OK** to close the Confirmation window.

19.3.10 How to Audit 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.

Table 19–3 describes what each validator checks for, as well as possible reasons why each validator might fail.

Table 19–3 Validator Descriptions

Validator	Checks for...	Validation may have failed because...	To correct...
Restrict by SetID Validator	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`.	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.	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`.
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 19–3 (Cont.) Validator Descriptions

Validator	Checks for...	Validation may have failed because...	To correct...
Available Data Sources Validator	Each data source view object specified for the tree structure must be accessible and all its primary key attributes should be valid.	■ Any of the specified data source view objects do not exist. ■ 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. ■ "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. ■ 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.	■ Consult the owning developer to correct the data source view object specified. ■ Consult the owning developer to correct the duplicate column in the registered primary keys. ■ Consult the owning developer to correct the primary keys of the data source view object specified. ■ Consult the owning developer to correct any mismatch in data types.
Column Flattened Table Name Validator	A valid "Column 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_CF`, or a custom table can be specified.	■ No table is specified in "Column Flattened Table" field 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_CF` table contains.	Consult the owning developer to correct the column flattened table definition.

Table 19–3 (Cont.) Validator Descriptions

Validator	Checks for...	Validation may have failed because...	To correct...
Restrict by Date Validator	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. This restriction does not apply when the flag is set to `No`.	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.	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`.
Tree Node Table Name Validator	A valid "Tree Node 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`, or a custom table can be specified.	▪ No table is specified in "Tree Node Table" 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.	Consult the owning developer to correct the tree node table definition.
Allow Node Level Security Validator	If "Allow Node Level Security" 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.	"Allow Node Level Security" flag is set as `No` for the tree structure, but one or more associated data sources have that flag set to `Yes`.	Consult the owning developer to correct the "Allow Node Level Security" flags in the tree structure and/or its data sources.

To audit a tree structure manually:

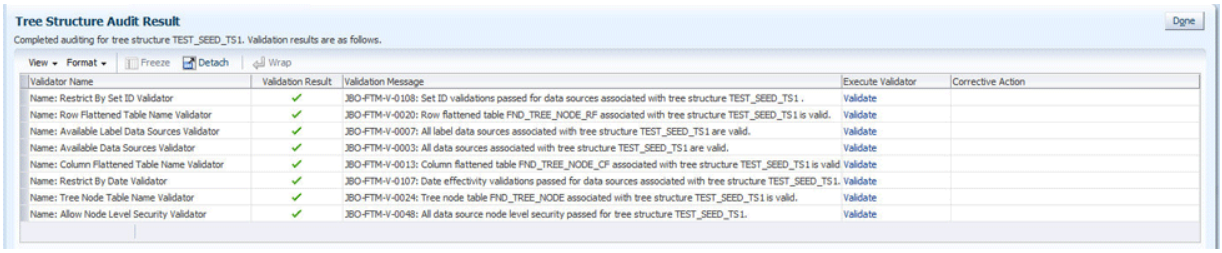
1. Select a tree structure.

   See Section 19.3.3, "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 Figure 19–30. The table displays a list of validations run against the selected tree structure.

Figure 19–30 Tree Structure Audit Result Page



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.

19.4 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 Section 19.5.8, "How to Audit Trees and Tree Versions."

19.4.1 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:

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.

Clicking **Advanced** enables you to perform an advanced search by specifying additional options, such as adding fields to search. You also can save your search criteria for future use.

19.4.2 How to Create a Tree

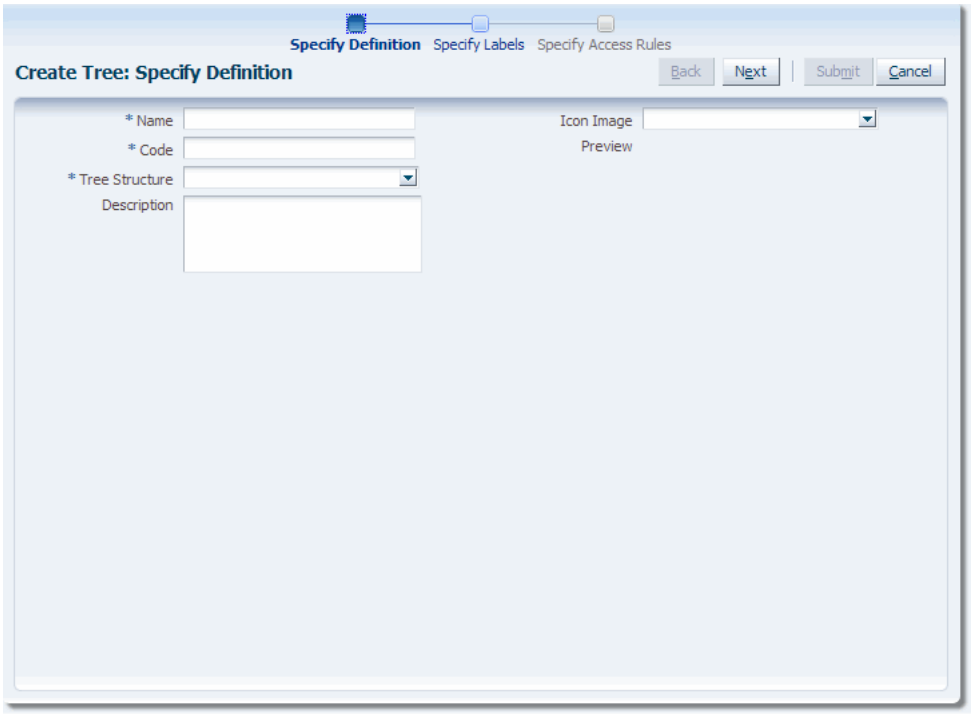
The following procedure explains how to create a tree. You also will need to create a tree version with a root node. For more information, see Section 19.5.1, "How to Create a Tree Version."

To create a tree:

1. Click **Create**, or choose **Create Tree** from the **Actions** dropdown menu.

   The Create Tree: Specify Definition page opens, as shown in Figure 19–31.

Figure 19–31 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 19–32 and Figure 19–33 show examples of both pages.

Figure 19–32 Create Tree: Specify Labels - No Labeling Scheme

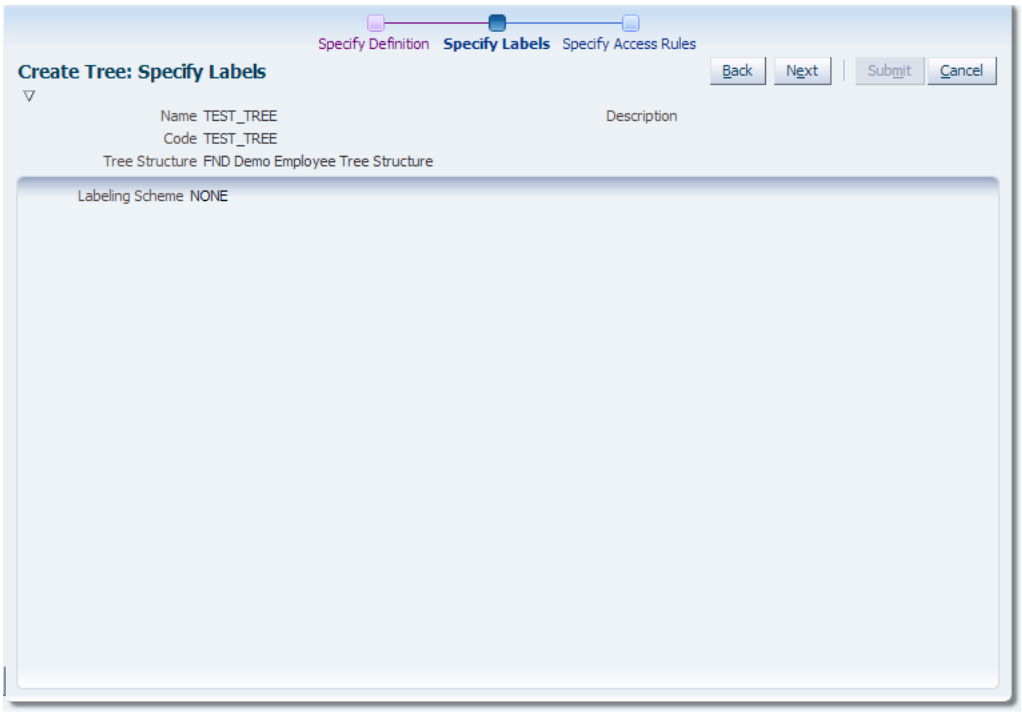
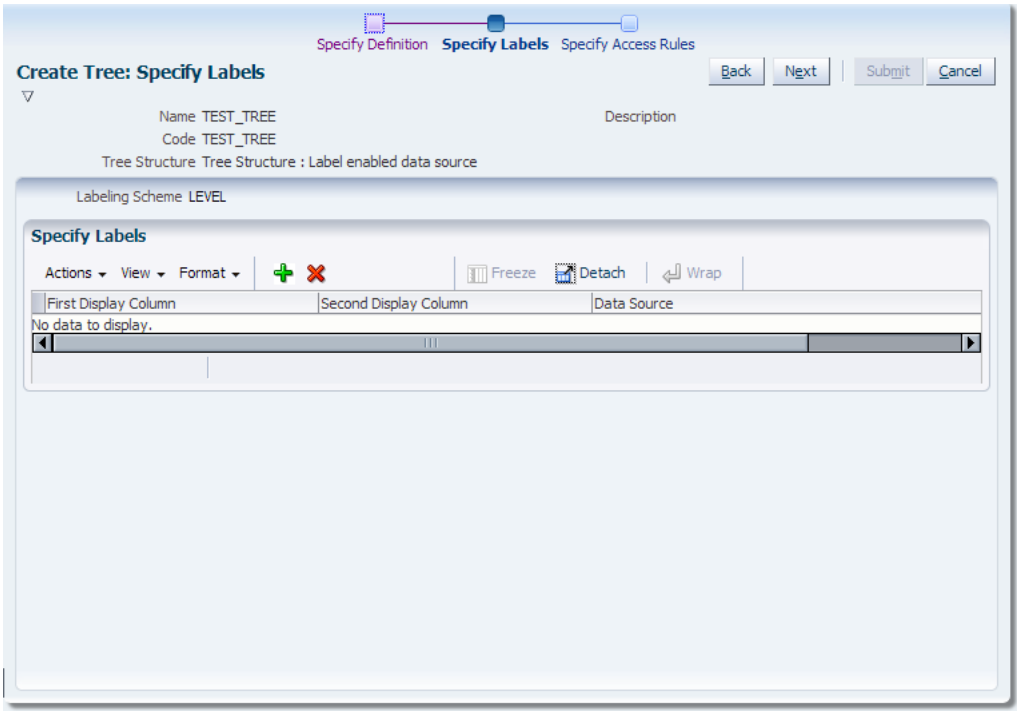


Figure 19–33 Create Tree: Specify Labels - Labeling Scheme Selected

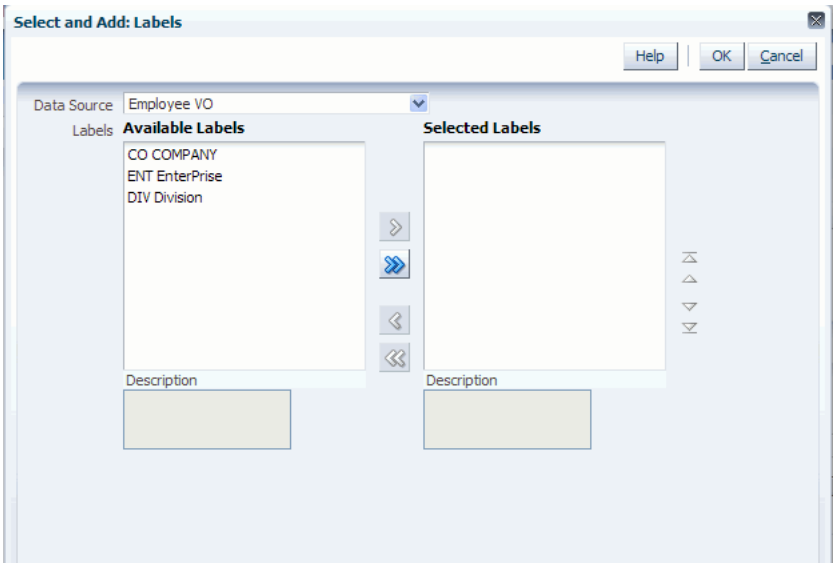


If the page shown in Figure 19–32 opens, click **Next** and skip to Step 11.

If the page shown in Figure 19–33 opens, click Add in the Specify Labels area.

The Select and Add: Labels window opens, as shown in Figure 19–34.

Figure 19–34 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 shuttle 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 Section 19.5.1, "How to Create a Tree Version."

19.4.3 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:

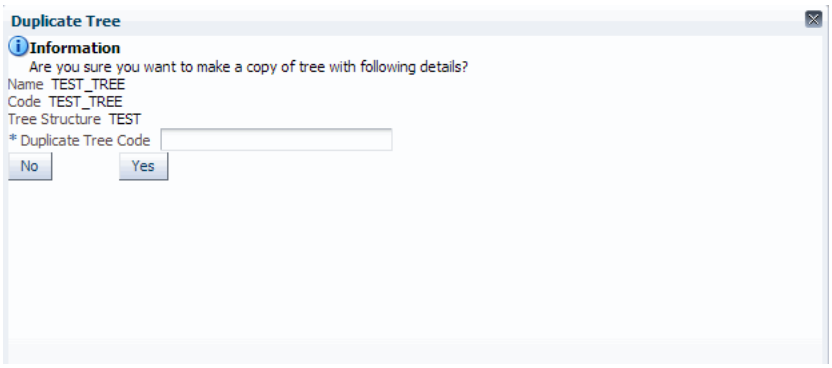
1. Select the tree you wish to duplicate.

   See Section 19.4.1, "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, as shown in Figure 19–35.

Figure 19–35 Duplicate Tree Window



3. Enter a duplicate tree code.
4. Do one of the following:
   - Click **No** to cancel the operation.
   - Click **Yes** to create the duplicate.
5. Click **OK** to close the Confirmation window.

19.4.4 How to Edit a Tree

When you edit an existing tree, you simply step through many of the same pages you used to create a tree.

To edit a tree:

1. Select the tree you wish to edit.

   See Section 19.4.1, "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

   Note: If you do change any parameter values, ensure that click **Actions > Save** before clicking **Next**.

4. Click **Next**.

   The Edit Tree: Specify Labels page opens. The page that displays depends on whether or not the tree structure used when 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 Section 19.4.2, "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 arow 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 Section 19.5.1, "How to Create a Tree Version," to create a new tree version.

19.4.5 How to Delete a Tree

When you delete a tree, you also delete the tree versions the tree contains.

To delete a tree:

1. Select the tree you wish to delete.

   See Section 19.4.1, "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. Confirm the deletion and click **OK**.

19.5 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
- Create a new record for a data source
- Duplicate
- Edit
- Delete
- Set tree version status
- Audit tree versions

19.5.1 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:

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.

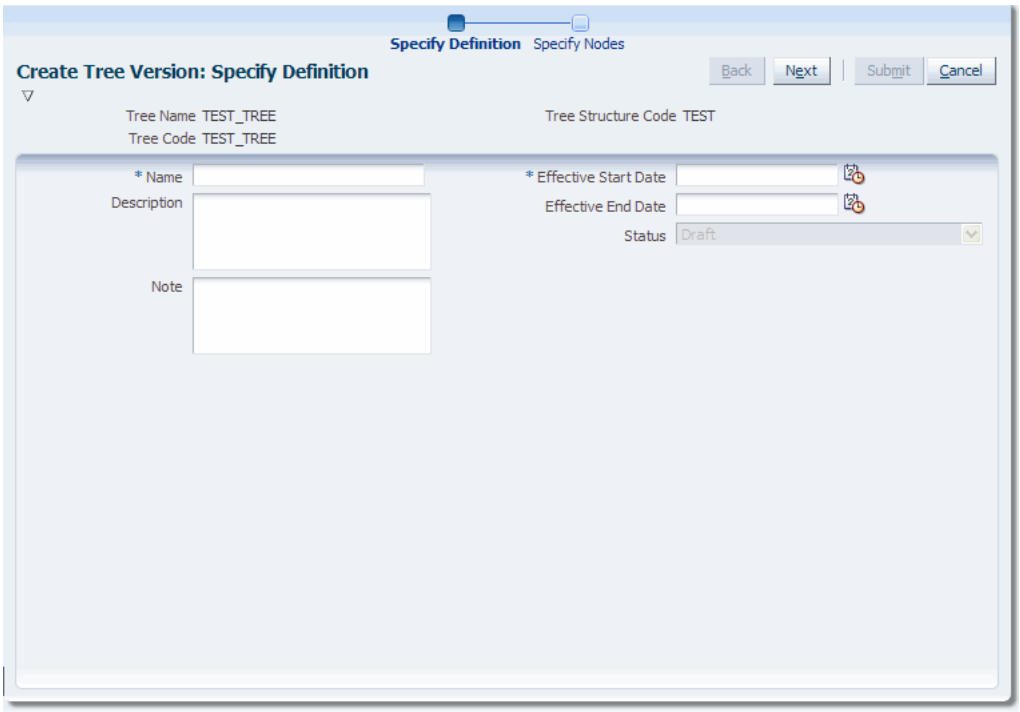
   See Section 19.4.1, "How to Search for a Tree," if the tree is not in the current Results list.

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, shown in Figure 19–36, opens.

Figure 19–36 Create Tree Version: Specify Definition Page



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.

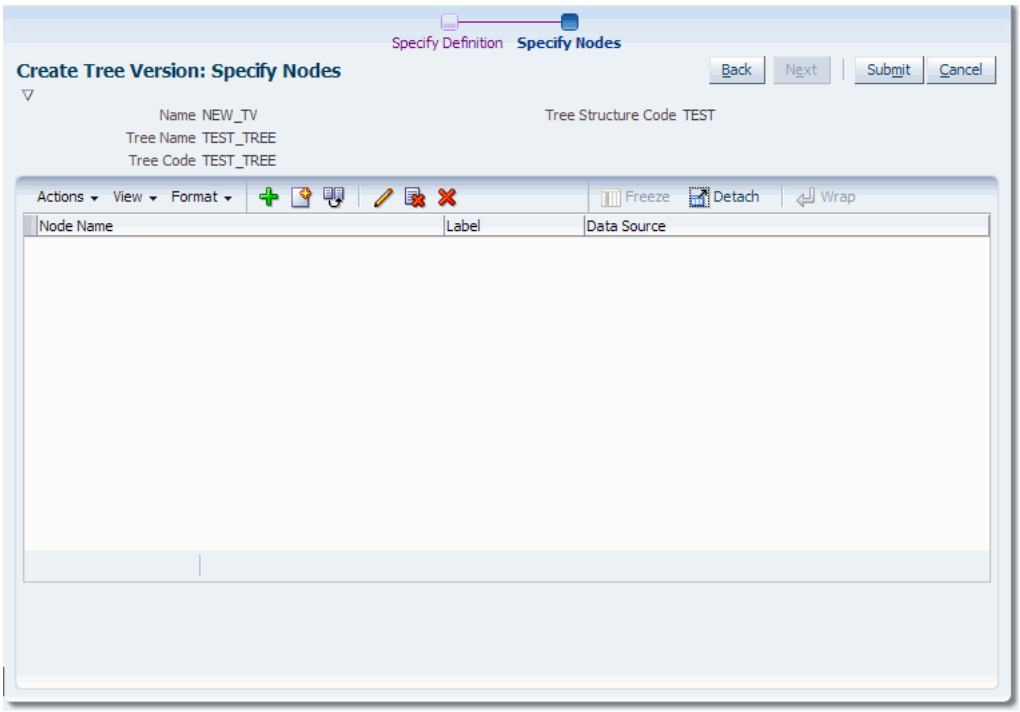
8. Click **Next**.

   A tree version with no nodes is created automatically at this point. Procedures for adding nodes to the tree version are described in.

9. Click **OK** to close the Confirmation window.

   The Create Tree Version: Specify Nodes page displays, as shown in Figure 19–37.

Figure 19–37 Create Tree Version: Specify Nodes Page

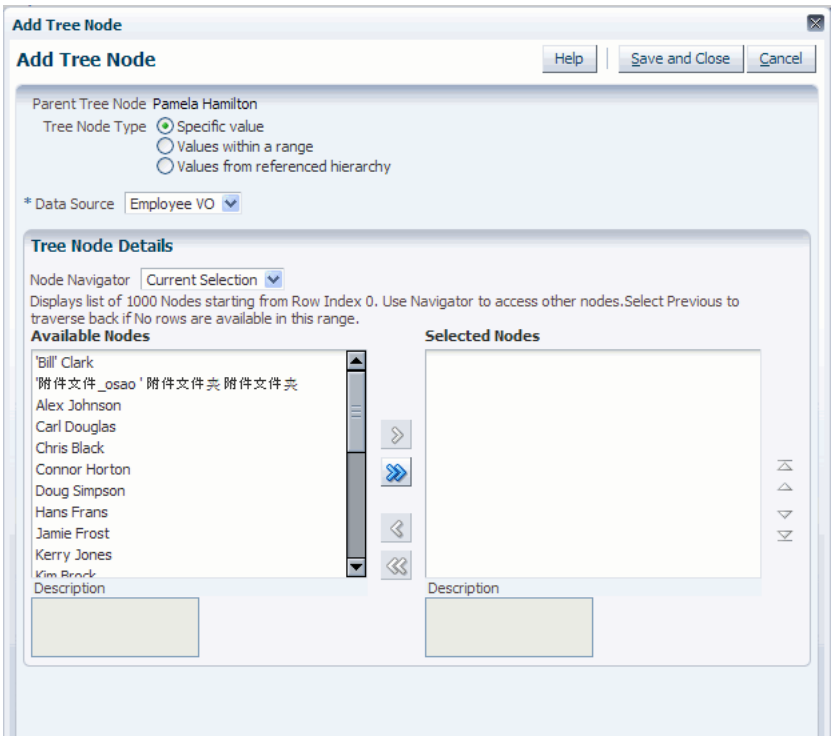


10. Do one of the following:

- Click **Add** to add a tree node.
- Click **Create** to create a new node in the data source and add it to the hierarchy. For more information, see Section 19.5.3, "How to Create a Record for a Data Source."

The Add Tree Node window opens, as shown in Figure 19–38.

Figure 19–38 Add Tree Node: Specific Values



Note: This is the default window that opens when adding a node.

---

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 Section 19.5.2.1, "How to Configure the Add Tree Node: Specific Values."

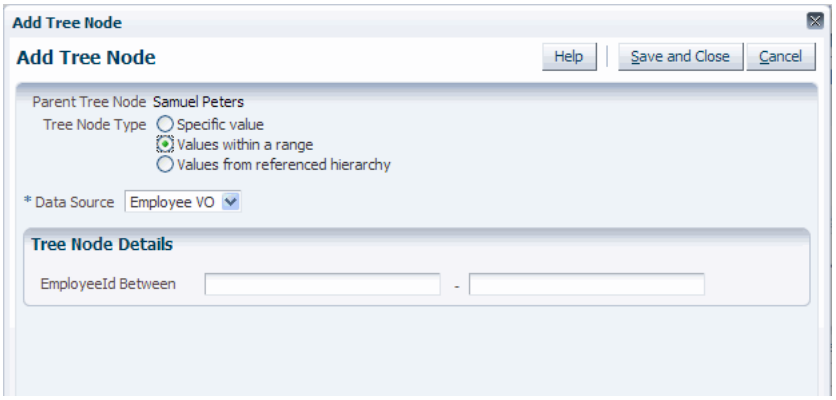
- **Values within a range** - Indicates that the node represents a range of values.

  Note: This option appears only if **Define Children By: Range** 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 window shown in Figure 19–40 replaces the default Add Tree Node window.

Figure 19–39 Add Tree Node: Values Within a Range

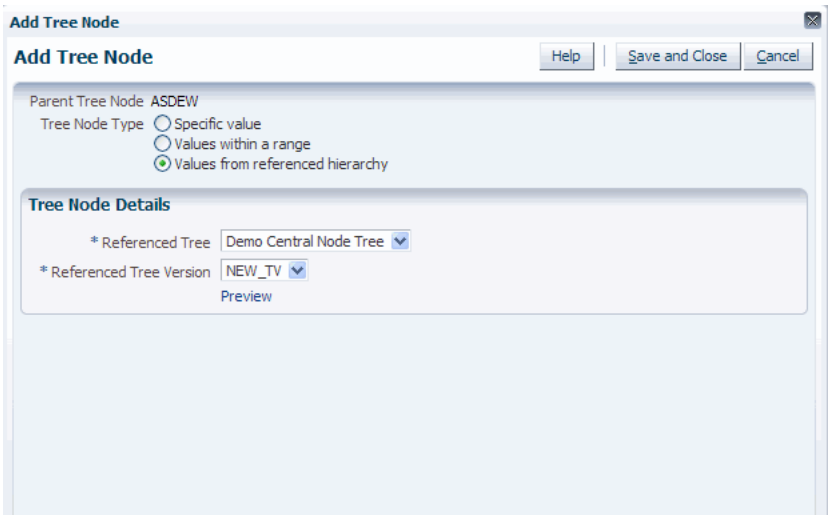


To configure this page's options, see Section 19.5.2.2, "How to Configure the Add Tree Node: Values Within a Range."

- **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 window shown in Figure 19–40 replaces the default Add Tree Node window.

Figure 19–40 Add Tree Node: Referenced Hierarchy



To configure referenced-tree options, see Section 19.5.2.3, "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.

19.5.2 How to Add 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, 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.

19.5.2.1 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:

This procedure assumes that the Add Tree Node window shown in Figure 19–38 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.

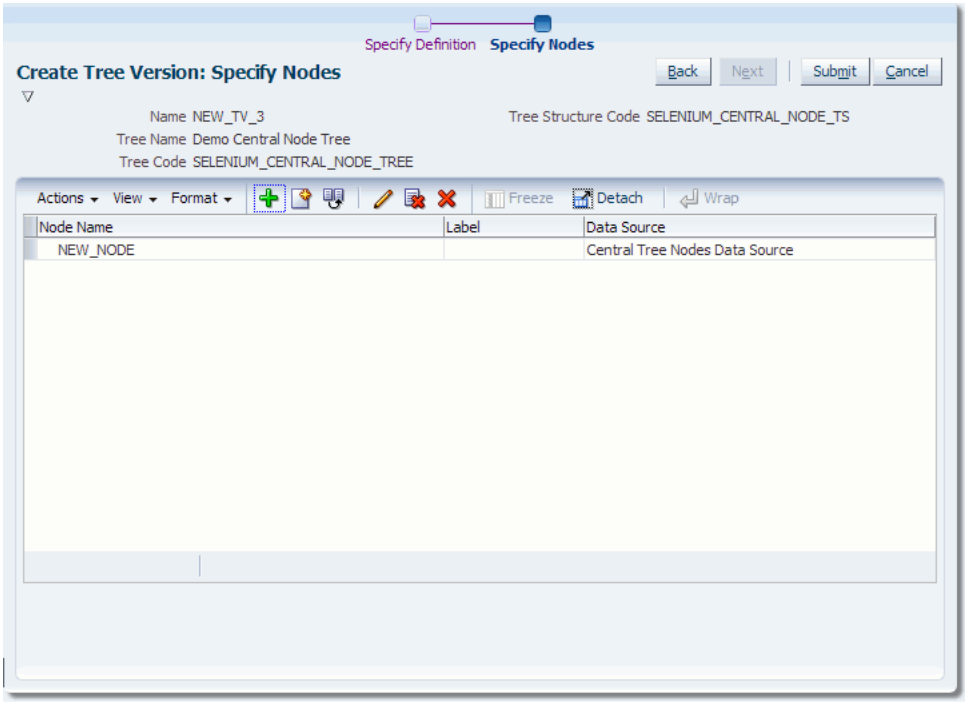
   Note: If the tree structure allows multiple root nodes to be selected, use the single or double Move arrows to move additional nodes.

6. Click **Save and Close**.

   The Add Tree Node window closes.

   Figure 19–41 shows the root node that has been created.

Figure 19–41 Root Node



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 Section 19.5.3, "How to Create a Record for a Data Source."
8. Review the data and click **Submit**.
9. Click **OK** to close the Confirmation window.

19.5.2.2 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:

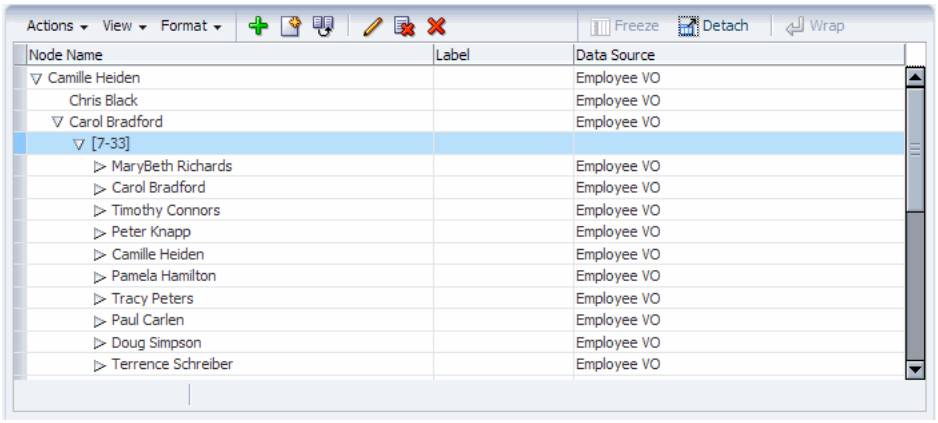
This procedure assumes that the Add Tree Node window shown inFigure 19–39 is open.

1. Select a data source.
2. Enter a range of values.
3. Click **Save and Close**.

   The Add Tree Node window closes.

   Figure 19–42 shows data based on a range of values.

Figure 19–42 Example of Range Data



4. Review the data and click **Submit**.
5. Click **OK** to close the Confirmation window.

19.5.2.3 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:

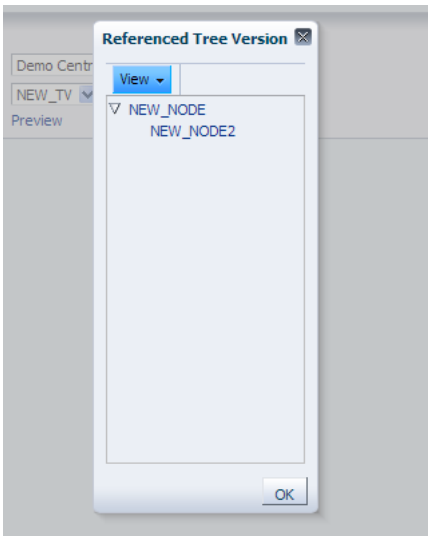
This procedure assumes that the Add Tree Node window shown in Figure 19–40 is open.

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. Figure 19–43 shows the window that displays. Confirm the data and click **OK** to close the window.

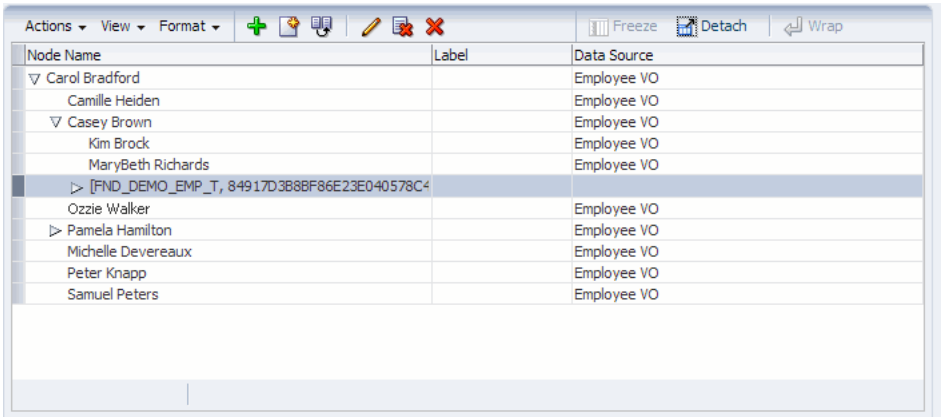
Figure 19–43 Preview Referenced Tree Version



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, as shown in Figure 19–44.

Figure 19–44 Referenced Node



5. Review the data and click **Submit**.
6. Click **OK** to close the Confirmation window.

19.5.2.4 How to Use 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.

19.5.2.5 How to Add a Node Using a Custom Search UI

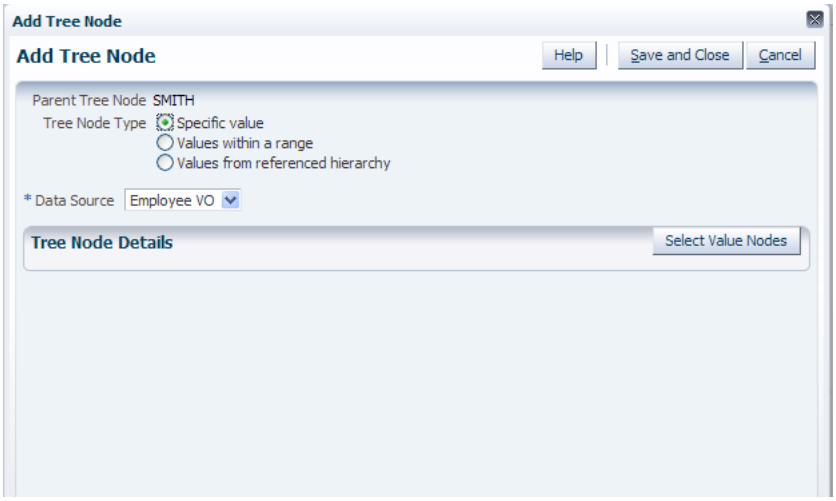
If the Search UI is not registered for a data source, the default behavior for all nodes displayed in the Add Tree Node window will be used. However, if you register your

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:

The procedures for adding a node using a custom Search UI are the same as those found in Section 19.5.2, "How to Add 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. An example of such a UI is shown in Figure 19–45. Note that **Select Value Nodes** has replaced the default Node Navigator, Available Nodes, and Selected Nodes options.

Figure 19–45 Custom Search UI: Specific Values



19.5.2.6 How to Edit a Tree Node

You can edit any existing tree node's details.

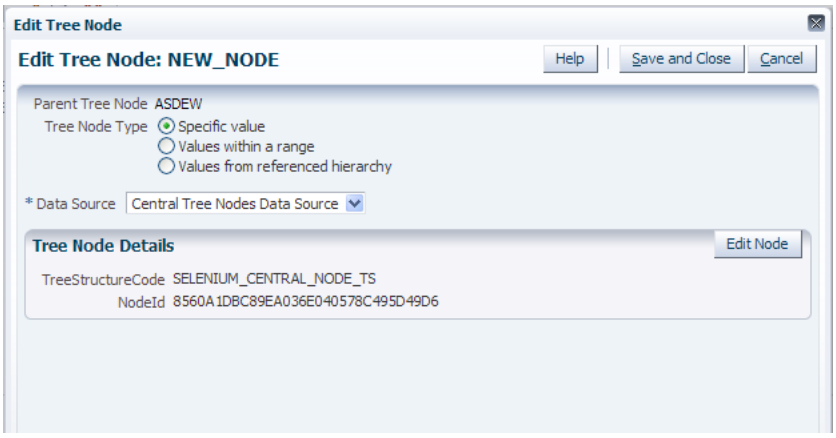
To edit a tree node:

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, shown in Figure 19–46, opens.

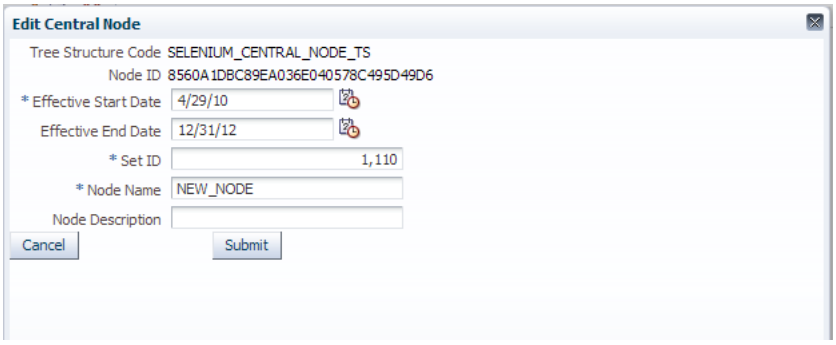
Note: The window opens with the default **Specific value** tree node type selected. You also can edit the node using the other tree node types. For more information, see Section 19.5.2, "How to Add Tree Nodes to a Tree Version."

Figure 19–46 Edit Tree Node: Specific Value



4. Select a data source and click **Edit Node**. The Edit Node window, shown in Figure 19–47, opens.

Figure 19–47 Edit Node Window



Note: The actual name of the window depends on the node being edited.

5. Edit the appropriate details and click **Save and Close**.

19.5.3 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:

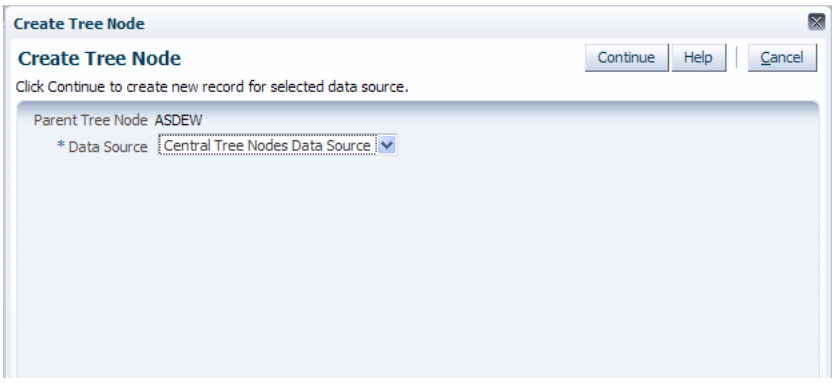
This procedure assumes the following:

- 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, as shown in Figure 19–48.

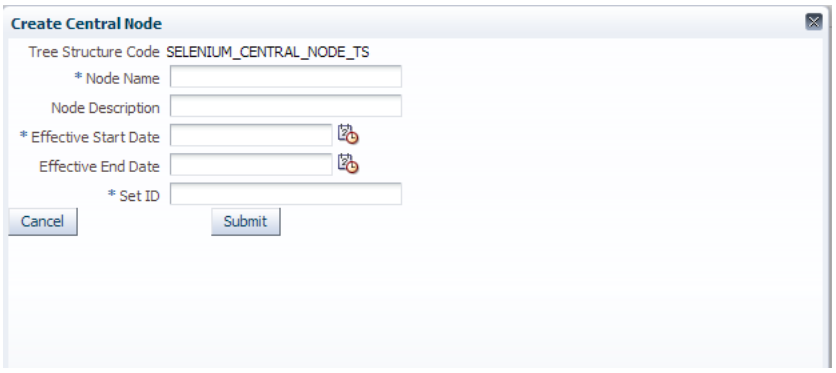
Figure 19–48 Create Tree Node Window



4. Select a data source and click **Continue**. The Create New Record window, shown in Figure 19–49, opens.

> **Note:** The actual name of the window depends on the node being created.

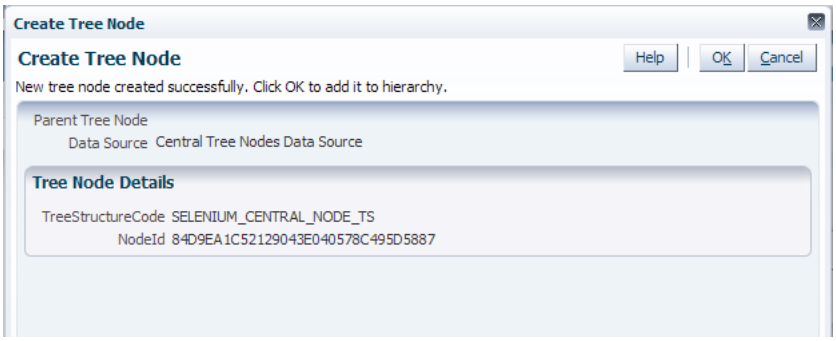
Figure 19–49 Create New Record for Data Source



5. Enter the appropriate information and click **Submit**.

   The Create Tree Node confirmation window, shown in Figure 19–50, displays.

Figure 19–50 Create Tree Node Confirmation



6. Click **OK** to close the window.

19.5.4 How to Duplicate a Tree Version

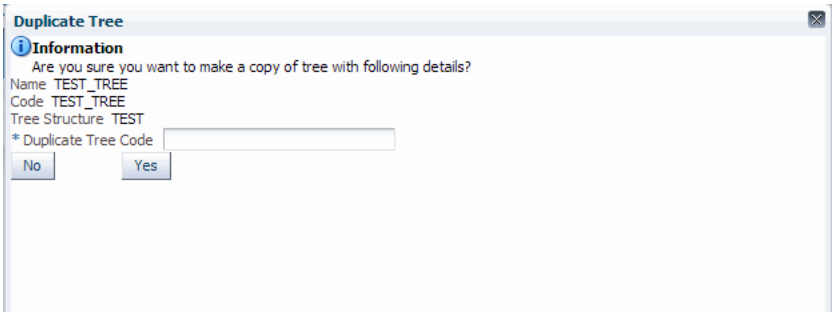
The following procedure explains how to duplicate a tree version.

To duplicate a tree version:

1. Select a tree version.
2. Click **Duplicate**, or choose **Duplicate** from the **Actions** dropdown menu.

   The Duplicate Tree Version window, shown in Figure 19–51, opens.

Figure 19–51 Duplicate Tree Version Window



3. Enter a duplicate tree version name.
4. Do one of the following:
   - Click **No** to cancel the operation.
   - Click **Yes** to duplicate the tree version.
5. Click **OK** to close the Confirmation window.

19.5.5 How to Edit a Tree Version

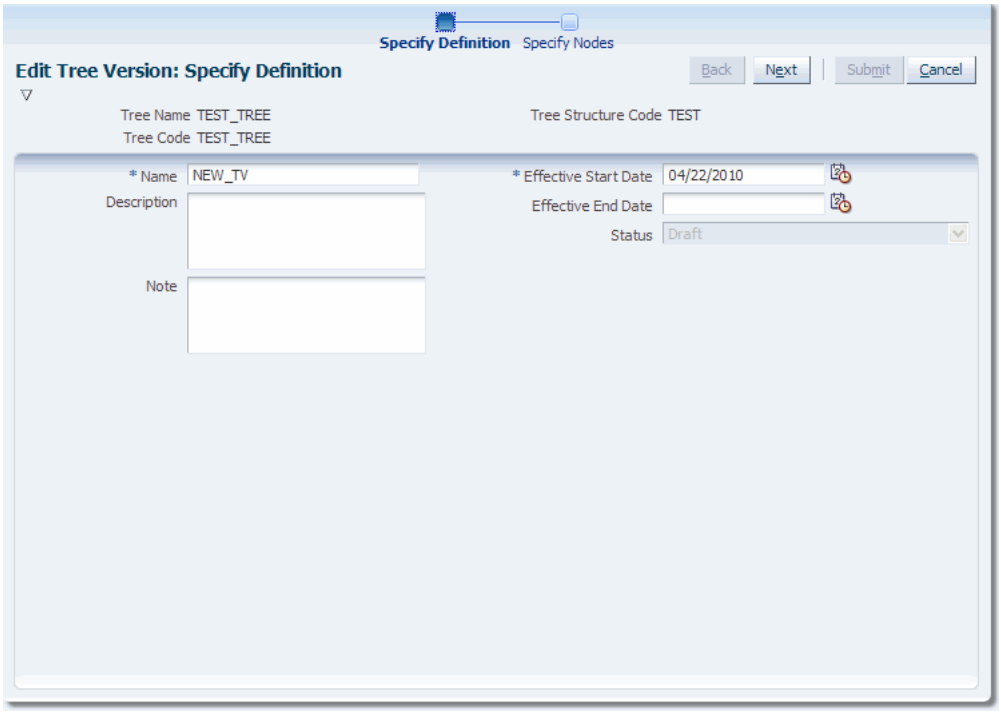
The following procedure explains to edit a tree version.

To edit a tree version:

1. Select a tree version.
2. Click **Edit**, or choose **Edit** from the **Actions** dropdown menu.

   The Edit Tree Version: Specify Definition page, shown in Figure 19–52, opens:

Figure 19–52 Edit Tree Version: Specify Definition Page



3. Use the steps in Section 19.5.1, "How to Create a Tree Version" as a guide to editing the tree version.

19.5.6 How to Delete a Tree Version

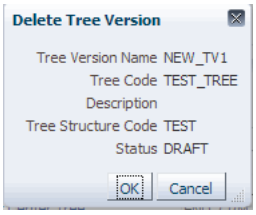
The following procedure explains how to delete a tree version.

To delete a tree version:

1. Select a tree version.
2. Click **Delete**, or choose **Delete** from the **Actions** dropdown menu.

   The Delete Tree Version warning window, shown in Figure 19–53, opens.

Figure 19–53 Delete Tree Version Warning Window



3. Do one of the following:
   - Click **Cancel** to cancel the operation.
   - Click **OK** to delete the tree version.
4. Click **OK** to close the Confirmation window.

19.5.7 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

In order 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. For more information, see Section 19.5.8, "How to Audit Trees and Tree Versions."

To set the status of a tree version:

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.

19.5.8 How to Audit 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

Table 19–4 describes what each validator checks for, as well as possible reasons why each validator might fail.

Table 19–4 Validator Descriptions

Validator	Checks for...	Validation may have failed because...	To correct...
Effective Date Validator	The effective start and end dates of the tree version should be valid.	Effective end date is set to a value that is not greater than effective start date.	Modify the effective start and/or end dates so that effective start date falls before effective end date.
Root Node Validator	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.	Allow Multiple Root Nodes flag has been set to `No` at the tree structure, but the tree version has multiple root nodes.	Modify the tree version so that there is exactly one root node.
Data Source Max Depth Validator	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).	Tree version has data at a depth greater than the specified depth limit on one or more data sources.	Modify the tree version so that all nodes are at a depth that complies with the data source depth limit.
Duplicate Node Validator	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.	Your tree version contains duplicate nodes with the same primary key.	Remove any duplicate nodes from the tree version.

Table 19–4 (Cont.) Validator Descriptions

Validator	Checks for...	Validation may have failed because...	To correct...
Available Node Validator	All the nodes in the tree version should be valid and available in the underlying data source.	▪ 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.	▪ Remove any orphaned nodes from the tree version. ▪ Correct any tree reference nodes so they reference existing tree versions.
Node Relationship Validator	All nodes should adhere to the relationships mandated by the data sources registered in the tree structure.	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.	Modify the tree version so that the nodes adhere to the same parent-child relationships as the data sources.
SetID Restricted Node Validator	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`.	The data source has Restrict Tree Node List of Values Based on SetID flag set to `Y`, but the tree version has nodes whose data source values belong to a different set than the tree.	Modify the tree version so that all nodes in the tree have data sources with SetID matching that of the tree.
Label Enabled Node Validator	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`.	The tree structure has a labeling scheme but the tree version has nodes without labels.	Assign labels to any nodes that do not have labels.

Table 19–4 (Cont.) Validator Descriptions

Validator	Checks for...	Validation may have failed because...	To correct...
Date Restricted Node Validator	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.	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).	Ensure that for all nodes in the tree version, they have date effectivity at least for the effective date range for the tree version.
Multiple Active Tree Version Validator	If Allow Multiple Active Tree Versions Flag 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.	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.	Make no more than one tree version Active within the same date range and set the others to Inactive or Draft.

Table 19–4 (Cont.) Validator Descriptions

Validator	Checks for...	Validation may have failed because...	To correct...
Range Based Node Validator	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.	There are range-based nodes from a data source that has Allow Range Children set to `N`.	Ensure that any range nodes in your tree version are from a data source that has Allow Range Children set to `Y`.
Terminal Node Validator	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.	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.	Modify the tree version so that all terminal nodes come from data sources with Allow Usage as Leaves set to `Y`.
Usage Limit Validator	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.	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.	Add nodes to the tree version for each data source value that is not yet present.

To run an immediate audit:

The following procedure assumes the Manage Trees and Tree Versions page is open.

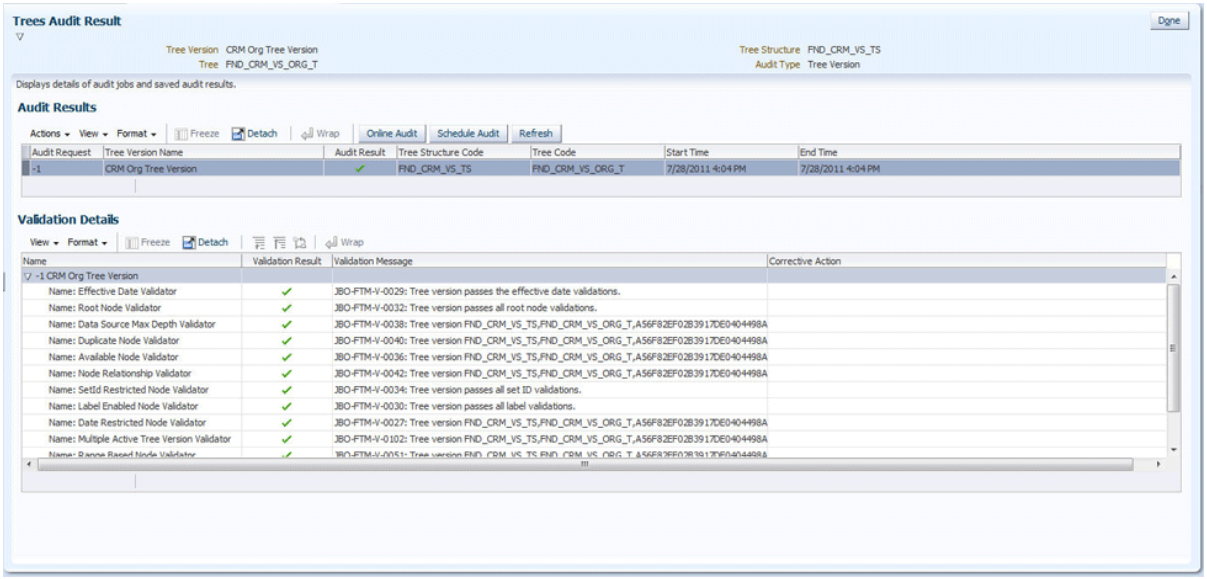
1. Select the tree or tree version you wish to audit.

   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, shown in Figure 19–54, opens.

Figure 19–54 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:

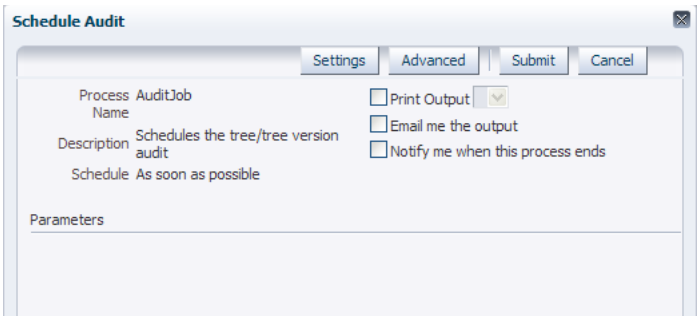
- **Name** - displays the names of the tree or tree version and the audit validators
- **Validation Result** - displays either a green check mark (success) or a red "X" (failure)
- **Validation Message** - when clicked, displays a validation message and a description
- **Corrective Action** - when clicked, opens the appropriate trees application task flow page, allowing you to fix a validation error

To schedule an audit:

1. Click **Schedule Audit**.

   The Schedule Audit window, shown in Figure 19–55, opens.

Figure 19–55 Schedule Audit Window



2. Do any of the following:
   - Configure the basic options.
   - Click **Advanced** to configure detailed schedule, output, and notification options.
   - Click **Settings** to configure language, territory, timezone, and other options.
3. Click **Submit** and then click **OK** to close the confirmation dialog.

To trigger an audit through a service API:

Use the service API shown in Example 19–1.

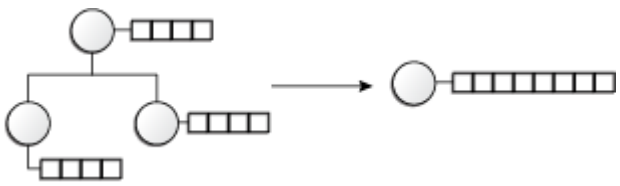
Example 19–1 Audit Service API

```
/**
   * 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 wheere 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)
```

19.5.9 How to Flatten 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. Figure 19–56 shows an example of a flattened tree structure.

Figure 19–56 Example of a Flattened Tree Structure



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 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.

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, it is common 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 run-time 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, as shown in Table 19–5.

Table 19–5 Rows and Columns in a Column-Flattened Table

Column 1	Column 2	Column 3
Region	Sales Division	Corporation

Usually, the numbers 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.

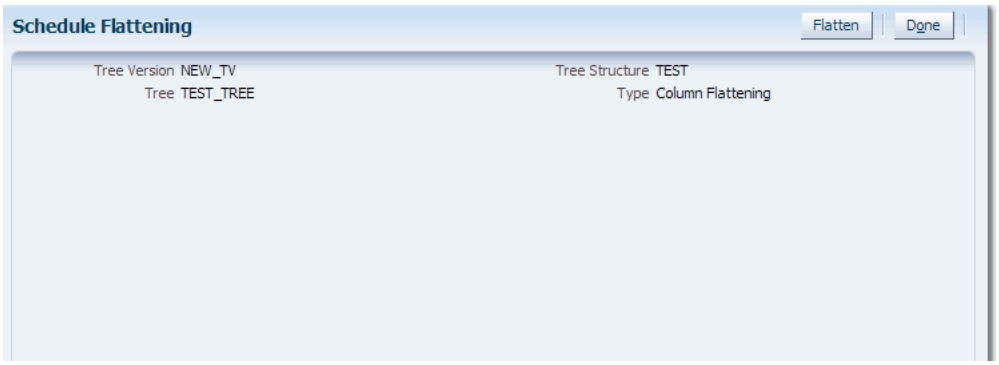
To flatten a row or column:

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 Schedule Flattening page, shown in Figure 19–57, displays.

Figure 19–57 Schedule Flattening Page



Note that the flattening process specified is the one that corresponds to your choice.

3. Click **Flattening**.
4. Click **OK** to close the Schedule Flattening confirmation window, shown in Figure 19–58.

Figure 19–58 Schedule Flattening Confirmation

5. Click **Done** to return to the Manage Trees and Tree Versions page.

19.6 Managing 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.

19.6.1 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
   - Short Name

2. Click **Search**.

   All labels matching your search criteria appear in the Results area of the page.

Clicking **Advanced** enables you to perform an advanced search by specifying additional options, such as adding fields to search. You also can save your search criteria for future use.

19.6.2 How to Create a Label

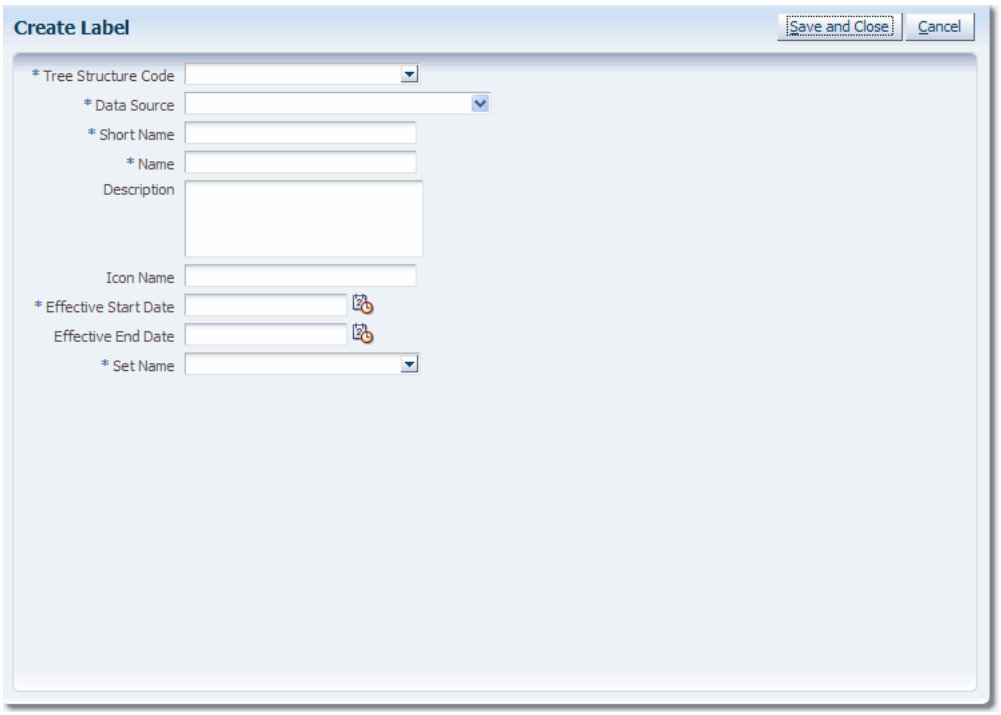
The following procedure explains how to create a new label.

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, as shown in Figure 19–59.

Figure 19–59 Create Label Page



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 ID.

10. Click **Submit** 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.

19.6.3 How to Edit a Label

The following procedure explains how to edit an existing label.

To edit a label:

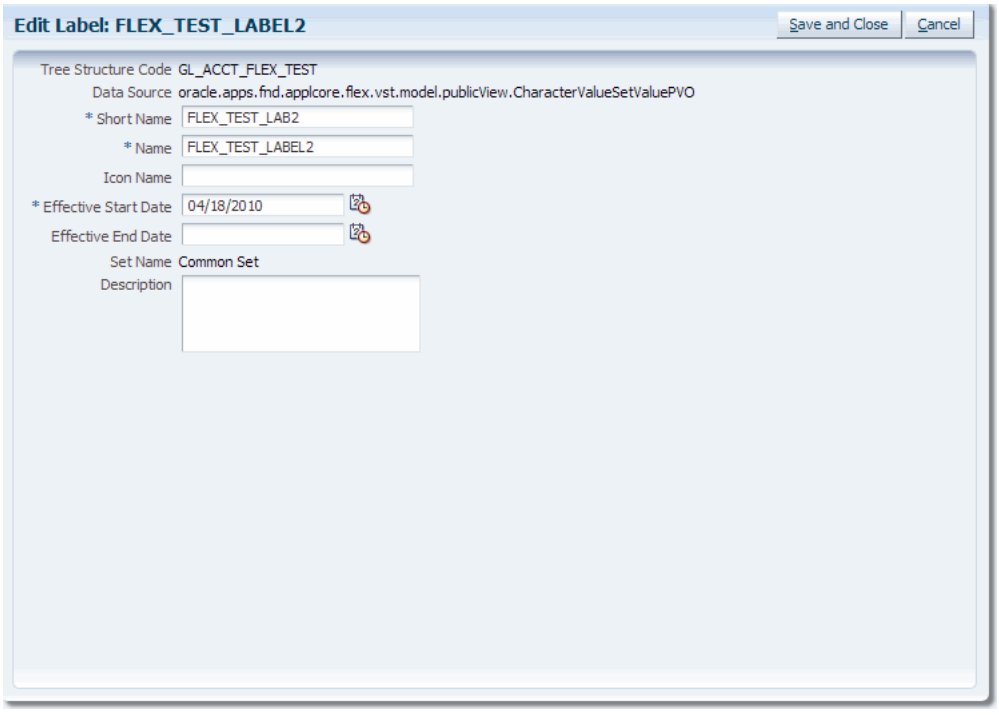
1. Select the label you want to edit.

   See Section 19.6.1, "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, shown in Figure 19–60.

Figure 19–60 Edit Label Page



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.

19.6.4 How to Delete a Label

The following procedure explains how to delete an existing label.

To delete a label:

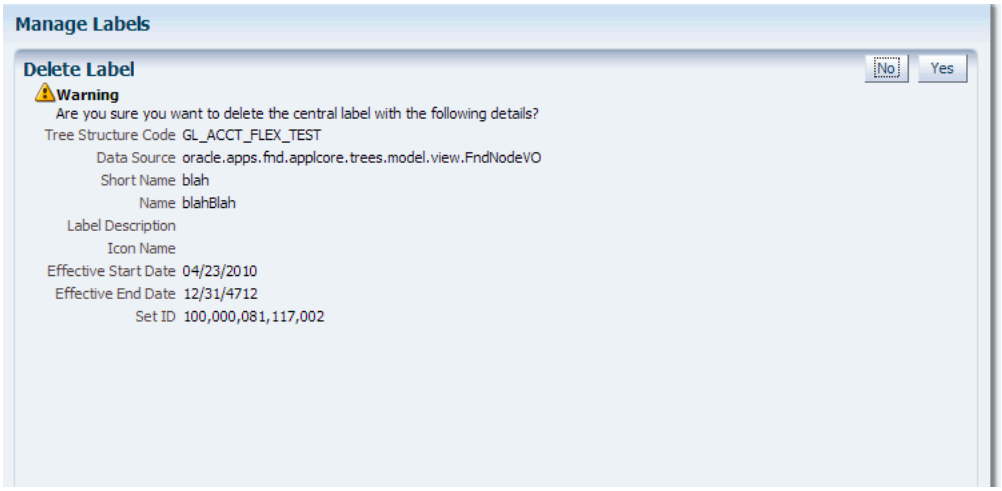
1. Select the label you want to delete.

   See Section 19.6.1, "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 shown in Figure 19–61 displays.

Figure 19–61 Delete Label Warning Page



3. Do one of the following:

   - Click **Yes** to delete the label.
   - Click **No** to cancel the operation.

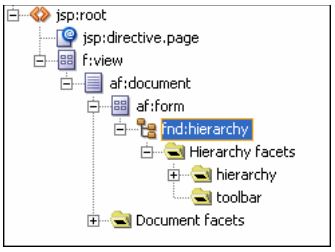
19.7 Using the Applications Hierarchy Component to Develop Applications

Now that you have worked with tree structures, trees, and tree versions, you can start to develop 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`.

Figure 19–62 shows an example of the Applications Hierarchy component in JDeveloper.

Figure 19–62 Applications Hierarchy Component



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.

19.7.1 How to Create a Tree Application

The following section explains how to create a tree application using the Applications Hierarchy component.

Before you begin:

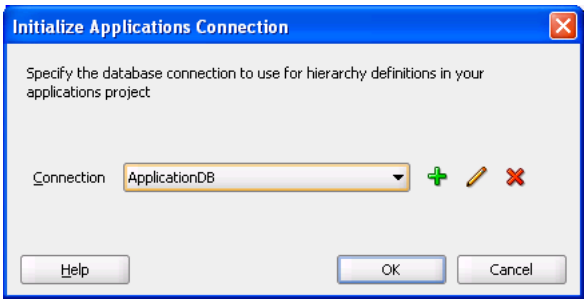
Create an application initialized for use with Oracle Middleware Extensions for Applications. For more information, see Chapter 2, "Setting Up Your Development Environment."

To create a Tree application:

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, as shown in Figure 19–63:

Figure 19–63 Initialize Applications Connection Window (1)

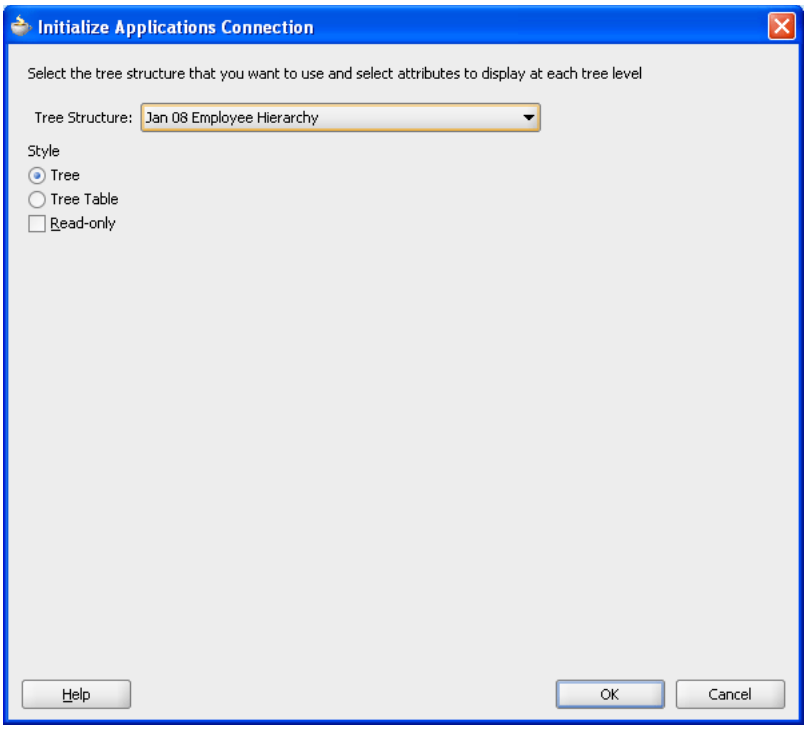


4. Choose a connection from the dropdown menu and click **OK**.

   If there is no existing connection, click **Add** to create a new one.

   A second Initialize Applications Connection window, shown in Figure 19–64, opens.

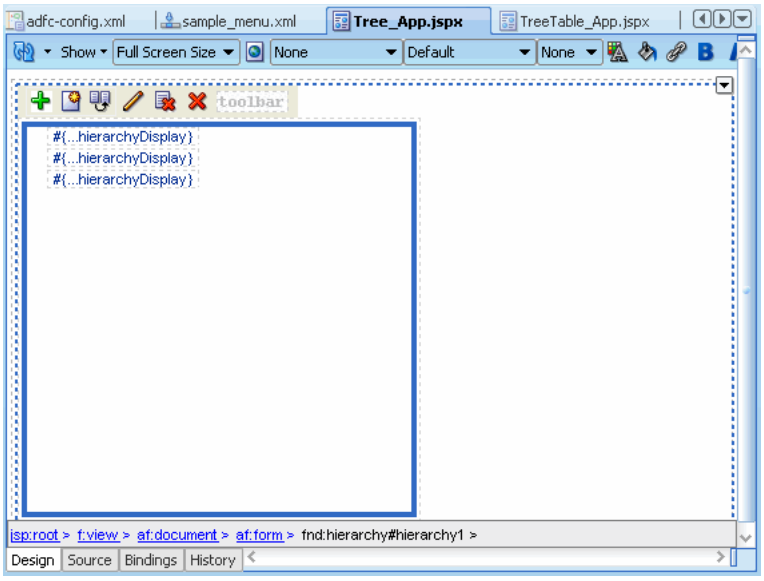
Figure 19–64 Initialize Applications Connection Window (2)



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, as shown in Figure 19–65.

Figure 19–65 Visual Editor with Tree



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. Figure 19–66 shows an example of a tree application.

Figure 19–66 Example of Tree Application Style

19.7.2 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:

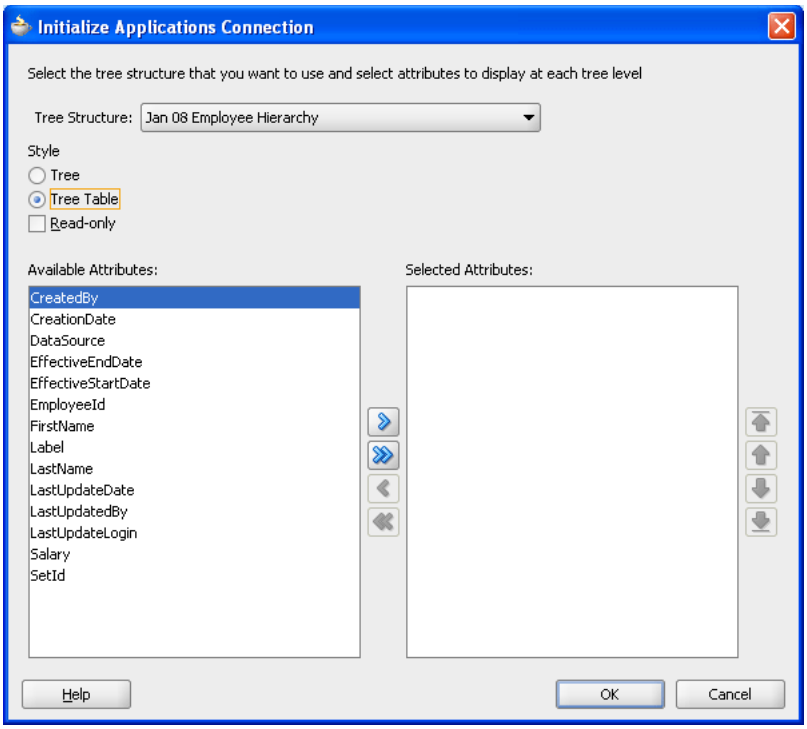
Create an application initialized for use with Oracle Middleware Extensions for Applications. For more information, see Chapter 2, "Setting Up Your Development Environment."

To create a Tree Table application:

1. Follow Steps 1 through 5 in Section 19.7.1, "How to Create a Tree Application."
2. Select **Tree Table**.

   The Initialize Applications Connection window redisplays with additional fields, as shown in Figure 19–67.

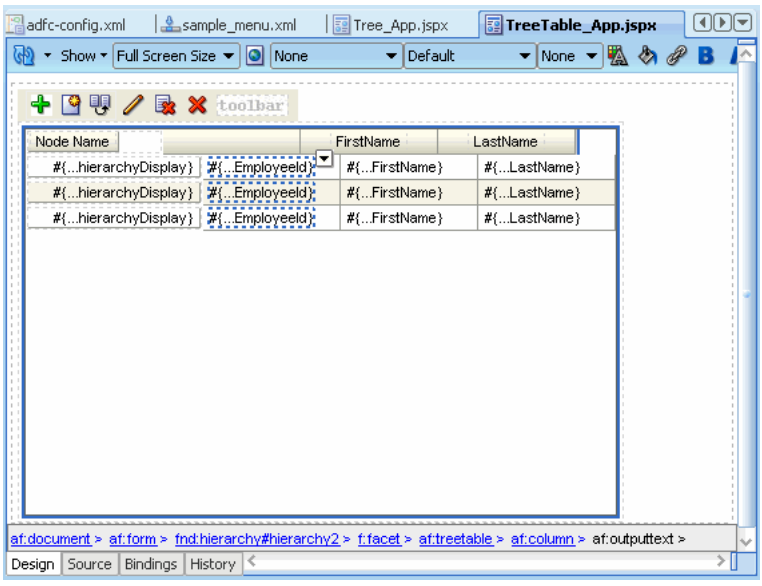
Figure 19–67 Initialize Applications Connection Window (3)



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, as shown in Figure 19–68.

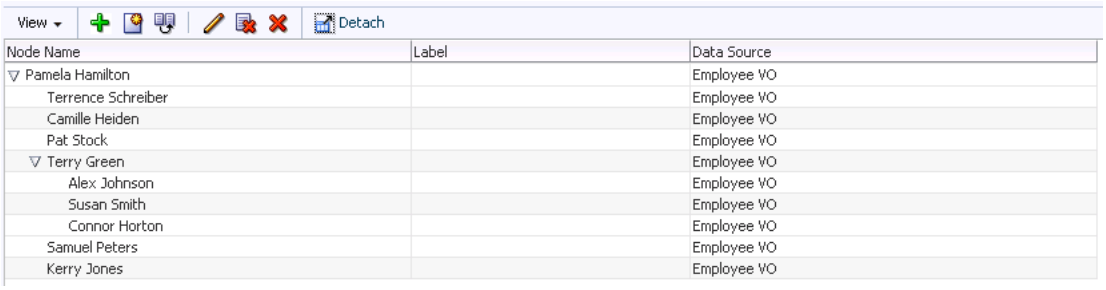
Figure 19–68 Visual Editor with Tree Table



6. Run the `.jspx` from either the Application Navigator or the visual editor.

A browser window opens and the application runs. Figure 19–69 shows an example of a tree table application.

Figure 19–69 Example of Tree Table Application Style



19.8 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.

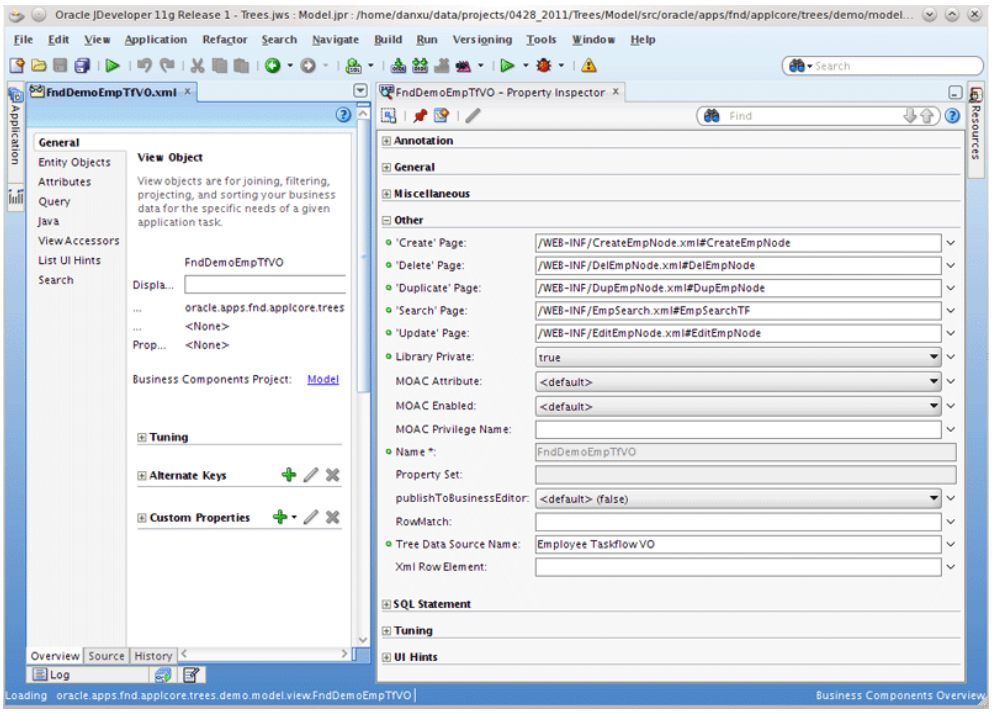
In addition, since removing a node affects only the data in the tree table, you do not need to create and register a custom task flow for this operation.

For more information about task flows, see "Getting Started with ADF Task Flows" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

19.8.1 Registering Custom Task Flows

You use the Property Inspector in Oracle JDeveloper to register custom task flows, as shown in Figure 19–70.

Figure 19–70 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 19–2 Search Task Flow for the Add Node Operation

```
<Properties>
  <SchemaBasedProperties>
    <fnd:SEARCH_PAGE Value="/WEB-INF/EmpSearch.xml#EmpSearchTF"/>
...
...
```

Example 19–3 Create Task Flow

```
<Properties>
  <SchemaBasedProperties>
    <fnd:CREATE_PAGE Value="/WEB-INF/CreateEmpNode.xml#CreateEmpNode"/>
...
...
```

Example 19–4 Duplicate Task Flow

```
<Properties>
  <SchemaBasedProperties>
    <fnd:DUPLICATE_PAGE Value="/WEB-INF/DupEmpNode.xml#DupEmpNode"/>
...
...
```

Example 19–5 Edit Task Flow

```
<Properties>
  <SchemaBasedProperties>
    <fnd:UPDATE_PAGE Value="/WEB-INF/EditEmpNode.xml#EditEmpNode"/>
...
...
```

Example 19–6 Delete Task Flow

```
<Properties>
  <SchemaBasedProperties>
    <fnd:DELETE_PAGE Value="/WEB-INF/DelEmpNode.xml#DelEmpNode"/>
...
...
```

19.8.2 Creating Custom Task Flows

This section discusses how to create custom task flows for the Search, Create, Duplicate, Edit, and Delete node operations.

19.8.2.1 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:

1. Define a task-flow parameter, searchHierParamBean, to pass values between the Hierarchy component and the registered Search task flow in pageFlowScope.

   ```
   <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:

   ```
   <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>
```

19.8.2.2 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:

1. Define a task-flow parameter, createHierParamBean, to pass values between the Hierarchy component and the registered Create task flow in pageFlowScope.

```
<input-parameter-definition>
  <name>createHierParam</name>
  <value>#{pageFlowScope.createHierParam}</value>
  <class>oracle.apps.fnd.applcore.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:

```
<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 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:

```
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:

```
<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>
```

19.8.2.3 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 the Duplicate task flow:

1. Define a task-flow parameter, dupHierParamBean, to pass values between the Hierarchy component and the registered Duplicate task flow in pageFlowScope.

```
<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:

```
<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.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:

```
List<String> pkValue = dupParam.getPkValue();
```

   In the `submit()` actionListener that maps to the *Submit* return activity, you pass the new primary keys back:

```
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:

```
<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}"/>
```

```
  <parameters>
</taskflow>
```

19.8.2.4 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:

1. Define a task-flow parameter, editHierParamBean, to pass values between the Hierarchy component and the registered Edit task flow in pageFlowScope.

```
<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:

```
<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.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:

```
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>
```

19.8.2.5 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:

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.applcore.trees.ui.managed.HierParamBean;
...

HierParamBean delParam =
(HierParamBean)AdfFacesContext.getCurrentInstance().getPageFlowScope().get
("delHierParam");
...
```

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:

```
List<String> pkValue = delParam.getPkValue();
```

In the `submit()` actionListener that maps to the *Submit* return activity, you specify the result:

```
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:

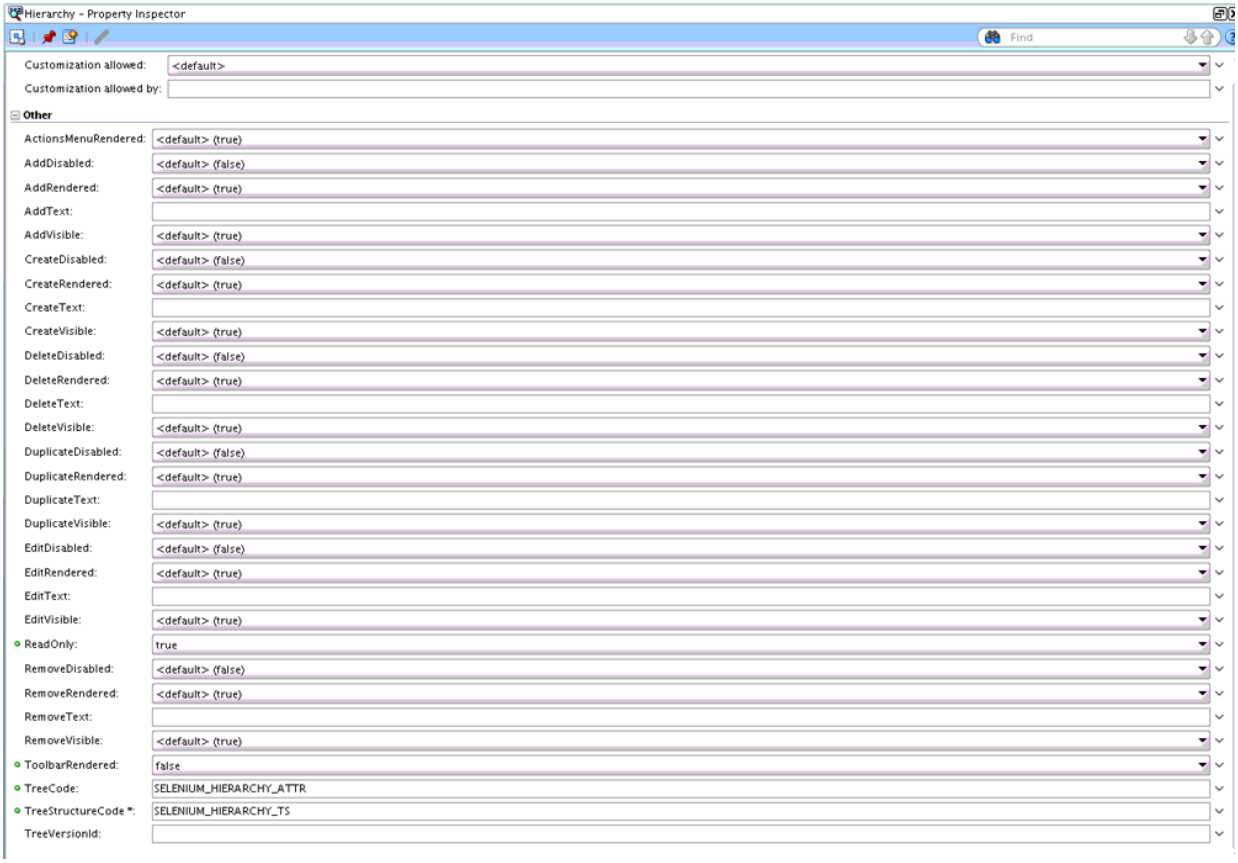
```
<taskFlow id="delTaskflow"
              taskFlowId="#{backingBeanScope.DelNodeBean.delTaskflow}"
              activation="deferred"
              Refresh="ifNeeded"
              xmlns="http://xmlns.oracle.com/adf/controller/binding">
  <parameters>
    <parameter id="delHierParam" xmlns="http://xmlns.oracle.com/adfm/uimodel"
               value="#{pageFlowScope.delHierParam}"/>
  </parameters>
</taskFlow>
```

19.9 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. Figure 19–71 shows the Hierarchy-Property Inspector.

Figure 19–71 Hierarchy-Property Inspector



The Hierarchy component has facets and properties, which are listed in Table 19–6 and Table 19–7.

Table 19–6 Hierarchy Facets

Facet	Description	Values
hierarchy	Holds ADF Tree or TreeTable	af:tree or af:treeTable
toolbar	Additional toolbar buttons to be added for custom use	ADF command toolbar buttons under an ADF toolbar

Table 19–7 Hierarchy Properties

Property	Description	Values
id	Unique identification number for hierarchy	string
rendered	Indicates if the hierarchy is rendered	boolean
readOnly	Indicates if the hierarchy will render in read-only mode	boolean
treeStructureCode	Code for the tree structure to be used for the hierarchy	string
tree Code	Code for the tree to be used for the hierarchy	string
treeVersionId	ID for the tree version to be used for the hierarchy	string
actionsMenuRendered	Controls if action menu needs to be rendered	boolean

Table 19–7 (Cont.) Hierarchy Properties

Property	Description	Values
toolbarRendered	Controls if toolbar needs to be rendered	boolean
addVisible	Controls if add action is visible	boolean
addRendered	Controls if add action is rendered	boolean
addDisabled	Controls if add action is disabled	boolean
addText	Custom text to be used for add action	string
createVisible	Controls if create action is visible	boolean
createRendered	Controls if create action is rendered	boolean
createDisabled	Controls if create action is disabled	boolean
createText	Custom text to be used for create action	string
duplicateVisible	Controls if duplicate action is visible	boolean
duplicateRendered	Controls if duplicate action is rendered	boolean
duplicateDisabled	Controls if duplicate action is disabled	boolean
duplicateText	Custom text to be used for duplicate action	string
editVisible	Controls if edit action is visible	boolean
editRendered	Controls if edit action is rendered	boolean
editDisabled	Controls if edit action is disabled	boolean
editText	Custom text to be used for edit action	string
removeVisible	Controls if remove action is visible	boolean
removeRendered	Controls if remove action is rendered	boolean
removeDisabled	Controls if remove action is disabled	boolean
removeText	Custom text to be used for remove action	string
deleteVisible	Controls if delete action is visible	boolean
deleteRendered	Controls if delete action is rendered	boolean
deleteDisabled	Controls if delete action is disabled	boolean
deleteText	Custom text to be used for delete action	string

registerTaskflow	Specifies whether to use task flow (`true`) or `.jspx` file (`false`).
	For example:
	`<fnd:hierarchy registerTaskflow="true"`
	`...`
	`</fnd:hierarchy>`
		boolean

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.

19.10 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 Section 19.9. 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.

19.11 Embedding 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:

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:

```
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:

```
Map pageFlow =
AdfFacesContext.getCurrentInstance().getPageFlowScope();

List<TreeNode> selectedNodes = (List<TreeNode>)
pageFlow.get("returnTreeNodes");
```

Figure 19–72 shows an example of a Tree Picker user interface.

Figure 19–72 Example Tree Picker

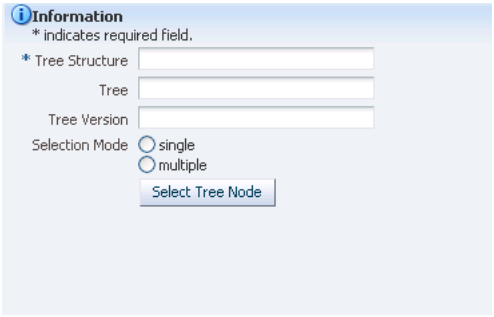
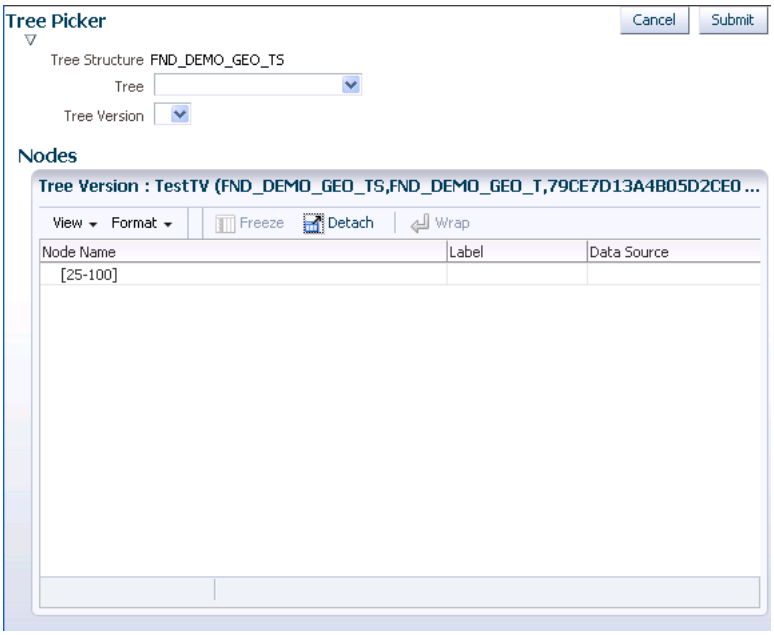


Figure 19–73 shows the results window that displays when you enter a tree-structure-code value and click **Select Tree Node**.

Figure 19–73 Tree Picker Results Window



19.12 Setting Bind Variables and View Criteria

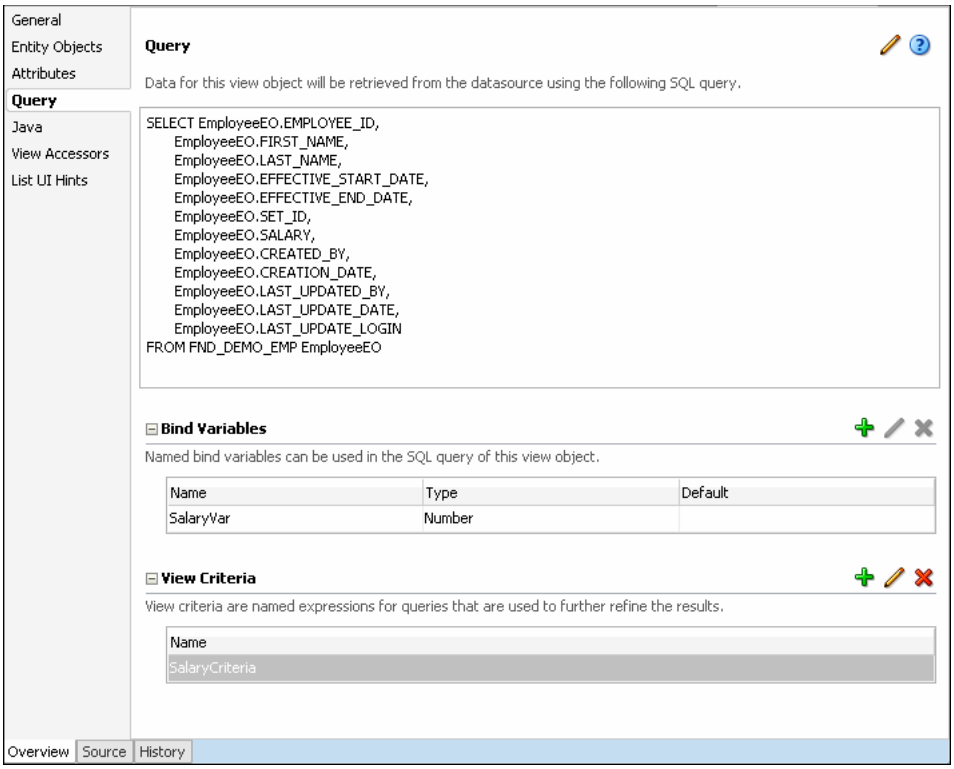
Every data source requires a view object. If the data source view object has bind variables and view criteria that tree management needs to apply, you must set them manually in JDeveloper.

19.12.1 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. Figure 19–74 shows the Bind Variables and View Criteria sections after the **Query** option is selected.

Figure 19–74 Bind Variables and View Criteria Settings



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.

19.13 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.

19.13.1 How to Use TreeStructureService

The TreeStructureService application module is defined in `oracle.apps.fnd.applcore.trees.service.applicationModule.TreeStr`

uctureService 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 Table 19–8.

Table 19–8 TreeStructureService APIs

API	Description
getTreeStructure	*FndTreeStructureVORow getTreeStructure(String treeStructureCode)* This API is used to retrieve the FndTreeStructureVORow corresponding to a particular tree structure code.
getRootDataSourceRels	*RowIterator getRootDataSourceRels(String treeStructureCode)* This API gets a row iterator over FndTsDataSourceRelVORow rows representing the root data sources of the given tree structure.
getAllTreeColumns	*List<AttributeDef> getAllTreeColumns(String treeStructureCode)* This API returns a list of VO attributes that are available for use from the various data sources associated with the tree structure. This is a cumulative list across all the data sources.
getAllDataSources	*oracle.jbo.RowIterator getAllDataSources(String treeStructureCode)* Returns a row iterator over FndTsDataSourceVORow rows corresponding to all data sources for the given tree structure.
getTreeNodeTable	*String getTreeNodeTable(String treeStructureCode)* Returns the name of the tree node table in use by a given tree structure.
duplicateTreeStructure	*void duplicateTreeStructure(String treeStructureCode, String duplicateTreeStructureCode)* This API is a Java front end to the PL/SQL API FND_TREE_UTILS.duplicate_tree_ structure and is used to duplicate a tree structure. It does not duplicate any underlying trees or tree versions associated with the tree structure.
deleteTreeStructure	*void deleteTreeStructure(String treeStructureCode)* 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).

19.13.2 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 Java Doc:

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 Table 19–9.

Table 19–9 TreeService APIs

API	Description
get TreeRows	*RowIterator getTreeRows(String treeStructureCode);* This API returns all trees associated with a given tree structure.
getTreeCodes	*List<String> getTreeCodes(String treeStructureCode);* This API returns a list of tree codes associated with a given tree structure.
findTree	*FndTreeVORow findTree(String treeStructureCode, String treeCode);* This API is used to find a specific tree given its tree structure code and tree code.
duplicateTree	*void duplicateTree(String treeStructureCode, String treeCode, String duplicateTreeCode);* 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.
deleteTree	*void deleteTree(String treeStructureCode, String treeCode);* 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.
getAllTreeVersions	*List<String> getAllTreeVersions(String treeStructureCode, String treeCode);* This API returns a list of all tree versions associated with a given tree.
getTreeVersions	*List<String> getTreeVersions(String treeStructureCode, String treeCode, Timestamp asOfDate);* This API returns a list of tree versions associated with a given tree as of a particular date.
getCurrentTreeVersions	*List<String> getCurrentTreeVersions(String treeStructureCode, String treeCode);* This API returns a list of tree versions associated with a given tree as of the current date.
findTreeVersion	*FndTreeVersionVORow findTreeVersion(String treeStructureCode, String treeCode, String treeVersionId);* This API is used to locate a specific tree version given its tree structure code, tree code and tree version ID.
duplicateTreeVersion	*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.
deleteTreeVersion	*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).
rowFlatten	*void rowFlatten(String treeStructureCode, String treeCode, String treeVersionId);* This API row-flattens a specific tree version.
columnFlatten	*void columnFlatten(String treeStructureCode, String treeCode, String treeVersionId);* This API column-flattens a specific tree version.

19.13.3 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 Java Doc:

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 Table 19–10.

Table 19–10 TreeNodeService APIs

API	Description
addValueTreeNode	*String addValueTreeNode(String treeStructureCode, String treeCode, String treeVersionId, String parentTreeNodeId, String dataSourceId, String pk1Value, String pk2Value, String pk3Value, String pk4Value, String pk5Value);* *String addValueTreeNode(String treeStructureCode, String treeCode, String treeVersionId, String parentTreeNodeId, String dataSourceId, String pk1Value, String pk2Value, String pk3Value, String pk4Value, String pk5Value, String treeLabelId);* 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.
addRangeTreeNode	*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.
addTreeTreeNode	*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.
deleteTreeNode	*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.
updateTreeNode	*void updateTreeNode(String treeStructureCode, String treeCode, String treeVersionId, 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 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.
moveTreeNode	*void moveTreeNode(String treeStructureCode, String treeCode, String treeVersionId, String treeNodeId, String destinationParentNodeId);* 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.

Table 19–10 (Cont.) TreeNodeService APIs

API	Description
findValueTreeNodes	*RowIterator findValueTreeNodes(String treeStructureCode, String treeCode, String treeVersionId, String[] pkValues);* This API is used to find all value tree nodes with the specified primary key.
findRangeTreeNodes	*RowIterator findRangeTreeNodes(String treeStructureCode, String treeCode, String treeVersionId, String[] pkStartValues, String[] pkEndValues);* This API is used to find all range tree nodes with the specified range.
findRefTreeNodes	*RowIterator findRefTreeNodes(String treeStructureCode, String treeCode, String treeVersionId, String refTreeCode, String refTreeVersionId);* This API is used to find all tree nodes that reference the specified tree version.

19.14 Advanced Topics

This section includes information about the following advanced topics:

- Tree data model
- PL/SQL APIs
- Incremental flattening
- Trees business events

19.14.1 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
- FND_TREE_FLATTENING_HISTORY

Views:

- FND_TREE_STRUCTURE_VL
- FND_LABEL_VL
- FND_TREE_VL
- FND_TREE_VERSION_VL
- FND_NODE_VL
- FND_TREE_VERSION_AUDIT_RES_VL

19.14.2 Using PL/SQL APIs

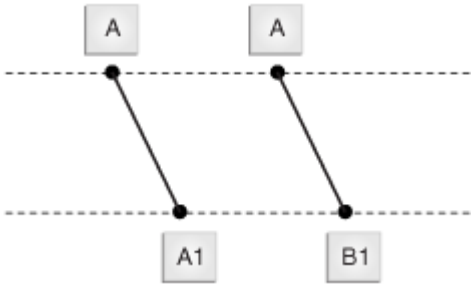
Tree Management provides public PL/SQL APIs to work with trees. You can find these APIs in the PL/SQL package FND_TREE_UTILS in the FUSION schema.

> **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.

19.14.3 Using Incremental Flattening

Incremental flattening optimizes the process by starting with the results of a previous flattening instead of flattening the data from scratch, as shown in Figure 19–75.

Figure 19–75 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.

19.14.3.1 How to Use 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.

Table 19–11 shows the contents of the FND_TREE_FLATENNING_HISTORY table.

Table 19–11 FND_TREE_FLATTENING_HISTORY

Column	Data Type	Nullable?
Tree_Structure_Code (Primary Key)	Varchar2(30)	No
Tree_Code (Primary Key)	Varchar2(30)	No
Tree_Version_ID (Primary Key)	Varchar2(32)	No
Process_Point	Timestamp(6)	No
Flattening_Type (Primary Key)	Varchar2(32)	No
Created_By	Varchar2(64)	No
Creation_Date	Timestamp(6)	No
Last_Updated_By	Varchar2(64)	No
Last_Update_Date	Timestamp(6)	No
Last_Update_Login	Varchar2(32)	Yes

19.14.3.2 How to Use 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

Table 19–12 shows the contents of the FND_TREE_LOG table.

Table 19–12 FND_TREE_LOG

Column	Data Type	Nullable?
Log_Entry_ID (Primary Key)	Varchar2(32)	No
Tree_Structure_Code	Varchar2(30)	No
Tree_Code	Varchar2(30)	No
Tree_Version_ID	Varchar2(32)	No
Operation_Type	Varchar1(32)	No
Created_By	Varchar2(64)	No
Creation_Date	Timestamp(6)	No
Last_Updated_By	Varchar2(64)	No
Last_Update_Date	Timestamp(6)	No
Last_Update_Login	Varchar2(32)	Yes

19.14.3.3 How to Use 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 table are references by a foreign key, **Log_Entry_ID**. This design helps save space and clearly organizes the information.

Table 19–13 shows the contents of the FND_TREE_LOG_PARAMS table.

Table 19–13 FND_TREE_LOG_PARAMS

Column	Data Type	Nullable?
Log_Entry_ID (Primary Key)	Varchar2(32)	No
Param_Name (Primary Key)	Varchar2(64)	No
Param_Value	Varchar2(100)	No
Created_By	Varchar2(64)	No
Creation_Date	Timestamp(6)	No
Last_Updated_By	Varchar2(64)	No
Last_Udpate_Date	Timestamp(6)	No
Last_Update_Login	Varchar2(32)	Yes

19.14.3.4 Flattening Rows

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 Section 19.14.3.4.1 and Section 19.14.3.4.2.

19.14.3.4.1 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).

19.14.3.4.2 DISTANCE This column indicates the distance between the node and its ancestor, which are 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. A simple example is shown in Figure 19–76.

In the FND_TREE_NODE table, DISTANCE is stored in the form of an adjacency list, as shown in Table 19–14.

Figure 19–76 Example of DISTANCE

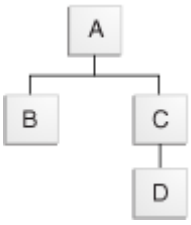


Table 19–14 Adjacency List

Node	Parent
A	Null
B	A
C	A
D	C

After flattening, DISTANCE is stored in the FND_TREE_NODE table, as shown in Table 19–15.

Note: The node ancestor also includes itself.

Table 19–15 Flattened FND_TREE_NODE Table

Node	Ancestor	Distance	IS_LEAF?
A	Null	1	N
A	A	0	N
B	Null	2	Y
B	A	1	Y
B	B	0	Y

Table 19–15 (Cont.) Flattened FND_TREE_NODE Table

Node	Ancestor	Distance	IS_LEAF?
C	Null	2	N
C	A	1	N
C	C	0	N
D	Null	3	Y
D	A	2	Y
D	C	1	Y
D	D	0	Y

To find the path from the root of D, the query would be the following:

```
select *
from fnd_tree_node_rf
where tree_node_id = D order by distance
```

19.14.3.5 Flattening Columns

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 Chapter 59, "Designing and Securing View Objects for Oracle Business Intelligence Applications.")

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, as shown in Figure 19–77.

Figure 19–77 Leaf-to-Root Order

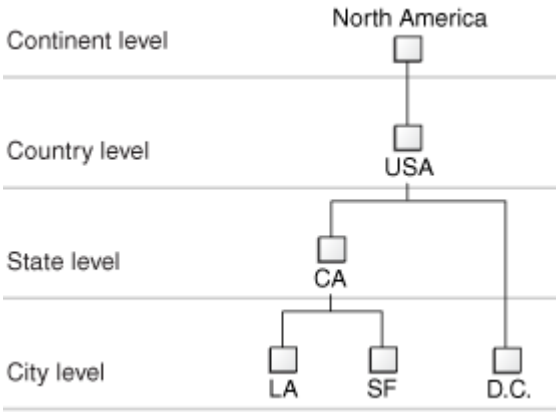


Table 19–16 shows the results after flattening.

Table 19–16 Column-Flattening Results

Dep0	Dep1	Dep2	Dep3	Dep4	...	Dep31
LA	CA	USA	North America	Null	...	Null
SF	CA	USA	North America	Null	...	Null
DC	Null	USA	North America	Null	...	Null

Note: All fields not containing nodes will be filled with Null.

19.14.4 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`.

Table 19–17 includes details of the create, update, and delete events.

Table 19–17 Trees Business Events

Entity	Event Name	Condition	Payload
Tree Structure	TreeStructureCreateEvent	Create	TreeStructureCode
Tree Structure	TreeStructureUpdateEvent	Update	TreeStructureCode
Tree Structure	TreeStructureDeleteEvent	Delete	TreeStructureCode
Tree	TreeCreateEvent	Create	TreeStructureCode, TreeCode
Tree	TreeUpdateEvent	Update	TreeStructureCode, TreeCode
Tree	TreeDeleteEvent	Delete	TreeStructureCode, TreeCode
Tree Version	TreeVersionCreateEvent	Create	TreeStructureCode, TreeCode, TreeVersionId
Tree Version	TreeVersionUpdateEvent	Update	TreeStructureCode, TreeCode, TreeVersionId
Tree Version	TreeVersionDeleteEvent	Delete	TreeStructureCode, TreeCode, TreeVersionId
Tree Node	TreeNode Created	New node added to tree (includes value, range and referenced tree nodes)	TreeStructureCode, TreeCode, TreeVersionId, TreeNodeId

Table 19–17 (Cont.) Trees Business Events

Entity	Event Name	Condition	Payload
Tree Node	TreeNode Deleted	Node removed from tree (includes value, range and referenced tree nodes)	TreeStructureCode, TreeCode, TreeVersionId, TreeNodeId
Tree Node	TreeNode Updated	Node updated in tree (includes value, range and referenced tree nodes)	TreeStructureCode, TreeCode, TreeVersionId, TreeNodeId
Tree Node	TreeNode Moved	Node moved within tree (includes value, range and referenced tree nodes)	TreeStructureCode, TreeCode, TreeVersionId, TreeNodeId

For more information, see "Using Business Events and the Event Delivery Network" in *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

20

Working with Localization Formatting

This chapter describes the Oracle Fusion Applications standards and guidelines for working with localization formatting.

20.1 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.

20.2 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, currency symbols, and so on.

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.

20.2.1 How to Format Currency Using Default Formatting Behavior

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.

To format currency using default formatting behavior:

1. In the Projects pane, select the view object.
2. In the Overview editor, click the **Attributes** tab, and select the attribute whose values you want to display as currency.
3. In the Property Inspector, select the **Applications** tab.
4. From the **Currency Type** dropdown list, select **True** to enable the attribute values to display as currency.

   A transient attribute called *AttributeName*`CurrencyCode` automatically displays, where *AttributeName* is the name of the attribute you wish to display as currency.
5. Use an expression to identify the currency code in the transient attribute.

   Currency codes are stored in the `FND_Currencies` table found in the Oracle Fusion Middleware Extensions for Applications (Applications Core) 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.

   For example, if the view object includes an attribute called `currencyCode`, you could use the expression shown in Example 20–1 to identify the currency code.

Example 20–1 Expression that Identifies the Currency Code

```
object.currencyCode
```

> **Note:** For more information about writing Groovy expressions, see the following link:
>
> `http://groovy.codehaus.org/`

6. To display currency values in a user interface, drag and drop the view object including the currency attribute onto the Java Server Faces (JSF) page.

> **Note:** In any user interface (UI) component associated with currency, locate the `convertNumber` element in Source view. Manually remove the attribute `type=currency` and save the file. This allows supporting non-ISO standard currencies.

20.2.2 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 Section 20.2.1, "How to Format Currency Using Default Formatting Behavior."

To format currency, overriding the default formatting behavior:

1. Follow the procedure described in Section 20.2.1, "How to Format Currency Using Default Formatting Behavior" 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 20–2.

Example 20–2 Bindings to the Number Converter with Specified Precision and Symbol Display

```
<af:outputText label="Amount" value="#{bindingToOrderTotal}">
<af:convertNumber pattern="#{fnd:currencyPatternWithPrecisionAndSymbol(
  bindingToAmountCurrencyCode, bindingToAttrNamePrecision,
  bindingToAttrNameCurrencySymbol)}" />
<af:outputText>
```

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.

20.2.3 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.

To immediately format currency using partial page rendering:

- For the UI component with a currency type value, do the following:
  a. Set the `autoSubmit` attribute to `true`.
  b. Set the `partialTriggers` attribute to the ID of the current UI component, as shown in Example 20–3.

Example 20–3 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>
```

20.2.4 What Happens When You Format Currency

When formatting currency, Applications Core automatically generates a transient view object attribute with the name *attributeName*`CurrencyCode`. 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 20–4.

Example 20–4 Bindings to the Number Converter

```
<af:outputText label="Amount" value="#{bindingToOrderTotal}">
<af:convertNumber type="currency"
 currencyCode="#{bindingToAmountCurrencyCode}"
 pattern="#{fnd:currencyPattern(bindingToAmountCurrencyCode)}" />
<af:outputText>
```

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.

20.2.5 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.

20.3 Formatting Dates and Numbers

Dates and numbers may be formatted differently, depending on local standards. For example, some regions may prefer to display the date first followed by the month, and use a decimal separator for numbers with four digits or more (for example, 1.234 to denote one thousand, two hundred and thirty four).

20.3.1 How to Format Dates and Numbers

The first of January 2010 can be displayed as follows:

- 1.1.10
- 1-1-10
- 1/1/10
- 1/1/2010
- 2010-01-01 (ISO standard)

Similarly, 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.

Before you begin:

Create an entity object and a view object with date and number fields, including any of the following attribute types:

- For numbers: `java.math.BigDecimal`, `java.lang.Integer`, or `java.lang.Long`.
- For dates: `java.sql.Date`.

To format dates and numbers:

Date and number values are displayed in the default formatting patterns as follows:

- `dateFormatPattern`: `dd-MMM-yyyy` (for example, `01-Jan-2010`)
- `numberFormatPattern`: `#,##0.###` (for example, `1,234.567`)

You can customize the date and number formatting patterns by editing the pattern attribute of the `af:convertDateTime` or `af:convertNumber` tags.

> **Note:** Date formatting is applicable only to view object attributes of data type `java.sql.Date`.

20.3.2 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 20–5 shows the bindings to the property `numericCodeFormatPattern`.

Example 20–5 Bindings to the numericCodeFormatPattern Property

```
<af:convertNumber pattern="#{applCorePrefs.numericCodeFormatPattern}"/>
```

The `intergerFormatPattern` property enables formatting an integer, as shown in Example 20–6. No fractions are printed.

Example 20–6 Bindings to the integerFormatPattern Property

```
<af:convertNumber pattern="#{applCorePrefs.integerFormatPattern}"/>
```

20.3.3 How to Format the Current Date

Applications often show the current date on the user interface. 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 therefore necessary to adjust the server date to the end user's local date.

Example 20–7 shows how to adjust the server date to the end user's local date.

Example 20–7 Adjusting the Server Date to the Local Date

```
public Date getCurrentLocalDate() {
    // Get the current date and time.
    long date = new java.util.Date().getTime();
    // Get the user preferred time zone from the ApplCore PreferencesBean.
    TimeZone uptz = TimeZone.getTimeZone(pb.getUPTZ());
    // Get the server time zone.
    TimeZone crtz = TimeZone.getDefault();
    // Calculate the time zone offset difference and return an adjusted date.
    int uptzoff = uptz.getOffset(date);
    int crtzoff = crtz.getOffset(date);
    int diff = uptzoff - crtzoff;
    return new Date(date+diff);
}
```

Example 20–8 shows the `af:inputDate` element on the page that sets the adjusted date. This correctly shows the default order date using the individual time zone of the end user.

Example 20–8 Populating the End-user Local Date on a Page

```
<af:inputDate value="#{localDateBean.currentLocalDate}"
                  label="#{bindings.OrderDate.hints.label}(Type: java.sql.Date)"
                  required="#{bindings.OrderDate.hints.mandatory}"
                  shortDesc="#{bindings.OrderDate.hints.tooltip}"
                  id="id1">
      <f:validator binding="#{bindings.OrderDate.validator}"/>
      <af:convertDateTime pattern="#{userPrefs.dateFormatPattern}"/>
</af:inputDate>
```

20.3.4 What Happens When You Format Dates and Numbers

All dates and number formatting patterns default to the formats described in Section 20.3.1, "How to Format Dates and 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 which binds to the `numberFormatPattern` property in the `applCorePrefs` managed bean as shown in Example 20–9.

Example 20–9 Bindings to the numberFormatPattern Property

```
<af:convertNumber pattern="#{applCorePrefs.numberFormatPattern}"/>
```

When dragging and dropping a view object containing an attribute of type `java.sql.Date`, Applications Core generates code which binds to the

dateFormatPattern property in the applCorePrefs managed bean as shown in Example 20–10.

Example 20–10 Bindings to the dateFormatPattern Property

```
<af:convertDateTime pattern="#{applCorePrefs.dateFormatPattern}"/>
```

Each code sample calls a date and number preference Applications Programming Interface (API), which obtains the property values from the Applications Session. The Applications Session, in turn, accesses the Oracle Fusion database and returns the correct date or number format.

20.3.5 What Happens at Runtime: How Dates and Numbers Are Formatted

At runtime, the bindings generated at design time are executed. Numbers and dates display according to user preferences for date and number formatting patterns (for example, 01/01/10 and 1,234.567).

20.3.6 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.

20.4 Formatting Time Zones

As opposed to date fields, date and time fields display times within a specific time zone. Date and time fields can be formatted to display in the user-preferred time zone or other applicable time zones.

JSF pages can display two types of dates: those that denote a day and time (January 1, 2008 01:00), and those that denote only the day (January 1, 2008). Date values that include a time component can be displayed with a relevant time zone value, for example, January 1, 2008 01:00 PST. Attributes that hold time zone data allow you to format and display a date-time value in the associated time zone.

Oracle database does not assume any time zone information when storing date information with the data types Date or Timestamp. The Java object types java.util.Date, java.sql.Date, and java.sql.Timestamp, on the other hand, use the UTC (Coordinated Universal Time) time zone. Oracle Java Database Connectivity (JDBC) reconciles the time zone values stored by these different data types by designating a *Java Virtual Machine* (JVM) default time zone for Oracle Date and Timestamp values. It then converts the values from JVM default time to UTC and vice versa.

According to development standards, developers must not change the JVM default time zone. This presents a challenge when developing an application that requires customized time zone values. The Date-Time Sensitive custom property enables you to properly handle time data while adhering to development standards. The default time zone is the corporate reporting time zone.

Note:

- Be sure to match the operating system's default zone on all middle tiers to the server time zone (the corporate reporting time zone).
- In accordance with Oracle Fusion Applications coding standards, time zone sensitive attributes must be of type `java.sql.Timestamp`. Date only fields must be of type `java.sql.Date`.

20.4.1 How to Format Time Zones

The Date-Time Sensitive custom property of a view object attribute is used to display time zone data in the relevant time zone—the user preferred time zone or other time zone. The user preferred time zone is retrieved from Oracle Applications Session, whereas the corporate reporting time zone is the default time zone of the server's operating system.

Note: Values are printed in UPTZ using the setting `timeZone="#{applCorePrefs.UPTZ}"` rather than the `UPTZPattern` attribute. The user preferences bean attribute `UPTZPattern` provides the user preferred formatting pattern for datetime values that do not explicitly print the time zone, such as 1/1/10 12:34 AM for an American user).

Values are printed in LETZ by setting the legal entity time zone to the `timeZone` attribute. The LETZPattern attribute in the user preferences bean is the user preferred formatting pattern for datetime values that explicitly print the time zone, such as 1/1/10 12:34 AM China Standard Time.

Before you begin:

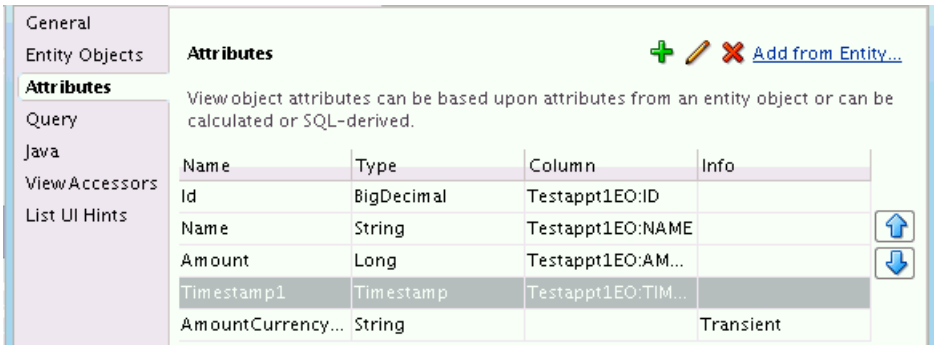
Create an entity object and a view object.

To format time zones:

1. In the Projects pane, select the view object.
2. In the Overview editor, click the **Attributes** tab, and select the attribute you want to configure as time sensitive.

   Figure 20–1 shows a selected attribute called `Timestamp1` in the Attributes tab.

Figure 20–1 Selecting the Time-Sensitive Attribute

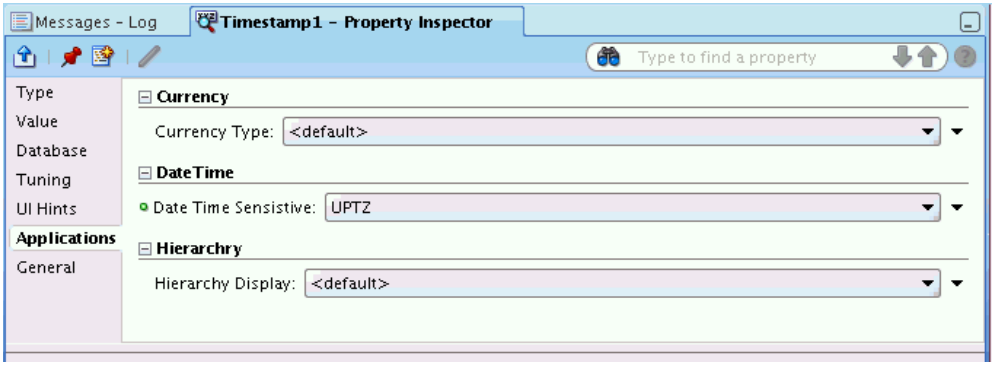


Note: Time zone sensitivity is applicable only to view object attributes of data type `java.sql.Timestamp`.

3. In the Property Inspector, select the **Applications** tab.
4. From the Date Time Sensitive dropdown list, select one of the following values:
   - default—Corporate Reporting Time Zone, to display the server time zone. This is the default value.
   - UPTZ—User Preferred Time Zone, to display the user preferred time zone.
   - LETZ—Legal Entity Time Zone, to display the time zone associated with a legal entity.

   Figure 20–2 shows the value **UPTZ** (User Preferred Time Zone) selected for the **DateTime** property of the `Timestamp1` attribute.

Figure 20–2 Selecting the Time Zone



Note: Do not use the Oracle ADF Faces date picker to enable end users to enter a date time value with a time zone. Instead, use a separate time zone selection UI component and use the date picker only for entering dates and times.

20.4.2 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 20–11 prints a datetime value in the user preferred format.

Example 20–11 Formatting in the User Preferred Datetime Pattern

```
<af:convertDateTime pattern="#{applCorePrefs.UPTZPattern}" />
```

Example 20–12 prints a datetime value in the user preferred format with seconds always included.

Example 20–12 Formatting in the User Preferred Datetime Pattern with Seconds

```
<af:convertDateTime pattern="#{applCorePrefs.DateFormatPattern}
```

```
#{applCorePrefs.timeFormatPatternWithSeconds}"/>
```

Example 20–13 prints a datetime value in the user preferred format without seconds.

Example 20–13 Formatting a Datetime Value in the User Preferred Datetime Pattern without Seconds

```
<af:convertDateTime pattern="#{applCorePrefs.DateFormatPattern}
#{applCorePrefs.timeFormatPatternWithoutSeconds}"/>
```

20.4.3 How to Format Invariant Time Zone Values

Some datetime 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 datetime 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 corporate reporting time zone, the server default, is applied implicitly.

20.4.4 What Happens When You Format Time Zones

Depending on the selected time zone, Applications Core generates tags as shown in Example 20–14 and Example 20–15.

Example 20–14 Default (Corporate Reporting) Time Zone

```
<af:outputText label="orderDateTime" value="#{bindingToOrderDateTime}">
<af:convertDateTime pattern="#{applCorePrefs.UPTZPattern}"/>
<af:outputText>
```

Example 20–15 User Preferred Time Zone

```
<af:outputText label="orderDateTime" value="#{bindingToOrderDateTime}">
<af:convertDateTime timeZone="#{applCorePrefs.UPTZ}"
 pattern="#{applCorePrefs.UPTZPattern}"/>
<af:outputText>
```

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.

20.4.5 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 `faces-config.xml`—retrieves the relevant formatting masks from Applications Session.

By default, date-time data may display as shown in Example 20–16.

Example 20–16 Date Time Data Format

```
1/1/2009 12:34 AM       for the pattern "M/d/yyyy hh:mm a"
```

20.4.6 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).

20.5 Formatting Numbers, Currency and Dates Using Localization Expression Language Functions

Expression Language functions provide an alternative to the formatting procedures described in Section 20.2, "Formatting Currency,", Section 20.3, "Formatting Dates and Numbers" and Section 20.4, "Formatting Time Zones."

20.5.1 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 20–17, the `dateValue` object must be converted to a `String` object by calling the `toString()` method.

Example 20–17 Binding a Date Object

```
af:formatNamed(bundle.NOTE_MESSAGE, 'BIRTHDAY', dateValue)
```

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.

20.5.1.1 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 20–18.

Example 20–18 formatNumber(java.lang.Number value) Function

```
fnd:formatNumber(java.lang.Number value)
```

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 20–19.

Example 20–19 Tag Produced by the Function fnd:formatNumber(java.lang.Number value)

```
<af:convertNumber pattern="#{applCorePrefs.numberFormatPattern}"/>
```

An additional Expression Language formatting function for numbers is shown in Example 20–20.

Example 20–20 formatNumber2(java.lang.Number value, int maxFractionDigit) Function

```
fnd:formatNumber2(java.lang.Number value, int maxFractionDigit)
```

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 20–21.

Example 20–21 Tag Produced by the Function fnd:formatNumber2(java.lang.Number value, int maxFractionDigit)

```
<af:convertNumber pattern="#{applCorePrefs.numberFormatPattern}"
           maxFractionDigits="your scale here"/>
```

20.5.1.2 Formatting Currency Using Expression Language Functions

Use the Expression Language function shown in Example 20–22 to format currency.

Example 20–22 fnd:formatCurrency(java.lang.Number currencyAmount, java.lang.String currencyCode) Function

```
fnd:formatCurrency(java.lang.Number currencyAmount,
            java.lang.String currencyCode)
```

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 20–23.

Example 20–23 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)}"/>
```

20.5.1.3 Formatting Dates Using Expression Language Functions

Use the Expression Language function shown in Example 20–24 to format dates.

Example 20–24 fnd:formatDate(java.util.Date dateValue) Function

```
fnd:formatDate(java.util.Date dateValue)
```

Returns the formatted date value based on the user preferred date format mask.

This function produces the tag shown in Example 20–25.

Example 20–25 Tag Produced by the Function fnd:formatDate(java.util.Date dateValue)

```
<af:convertDateTime pattern="#{applCorePrefs.dateFormatPattern}"/>
```

Use the Expression Language function shown in Example 20–26 to format date time values.

Example 20–26 fnd:formatDateTime(java.util.Date dateTimeValue) Function

```
fnd:formatDateTime(java.util.Date dateTimeValue)
```

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 20–27.

Example 20–27 Tag Produced by the Function fnd:formatDateTime(java.util.Date dateTimeValue)

```
<af:convertDateTime type="both" timeZone="#{applCorePrefs.UPTZ}"
             pattern="#{applCorePrefs.UPTZPattern}"/>
```

Use the Expression Language function shown in Example 20–28 to format date time values with user formatting masks and the user-specified time zone.

Example 20–28 fnd:formatDateTimeTZ(java.util.Date dateTimeValue, java.util.TimeZone timeZone) Function

```
fnd:formatDateTimeTZ(java.util.Date dateTimeValue,
                        java.util.TimeZone timeZone)
```

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 20–29.

Example 20–29 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}"/>
```

20.5.2 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 20–30.

Example 20–30 Tag Produced by Expression Language Date Formatting Function

```
<af:convertDateTime pattern="#{applCorePrefs.dateFormatPattern}"/>
```

20.5.3 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 Section 20.2, Section 20.3 and Section 20.4.

20.6 Configuring 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, in order 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.

20.6.1 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 20–1 lists the Oracle Fusion Applications session NLS attributes, the profile used to set each attribute and the possible values for session attributes.

Table 20–1 Session NLS Attributes

Session Attribute	Profile	Values	Comments
LANGUAGE	FND_LANGUAGE	`select DESCRIPTION, LANGUAGE_TAG from FND_LANGUAGES_B where INSTALLED_FLAG in ('I', 'B');` 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.	
NLS_LANG		Represents the two letter language code, which is derived using the LANGUAGE attributes. This value is derived rather than explicitly set.	
NLS_LANGUAGE		Represents the NLS language, which is derived from the LANGUAGE attribute. This value is derived rather than explicitly set.	
NLS_SORT	FND_NLS_SORT	`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);`	Setting this attribute results in an altered session in the database for the NLS_SORT database attribute.
DATE_FORMAT	FND_DATE_FORMAT	`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);`	
TIME_FORMAT	FND_TIME_FORMAT	`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);`	

Table 20–1 (Cont.) Session NLS Attributes

Session Attribute	Profile	Values	Comments
GROUPING_ SEPARATOR	FND_GROUPING_ SEPARATOR	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);	
DECIMAL_ SEPARATOR	FND_DECIMAL_ SEPARATOR	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);	
CURRENCY	FND_CURRENCY	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);	This attribute specifies the preferred currency code. It has no corresponding database attribute.
TERRITORY	FND_TERRITORY	select TERRITORY_SHORT_NAME, TERRITORY_CODE from FND_ TERRITORIES_VL where ENABLED_ FLAG = 'Y';	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.
TIMEZONE	FND_TIMEZONE	select TIMEZONE_CODE, NAME from FND_TIMEZONES_VL where ENABLED_ FLAG = 'Y';	This attribute specifies the preferred time zone value.
CLIENT_ENCODING	FND_CLIENT_ ENCODING	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);	

Table 20–2 lists language and territory values used with NLS attributes.

Table 20–2 Language and Territory Values

LANGUAGE_TAG	LANGUAGE_CODE	LANGUAGE_ID	NLS_LANGUAGE	NLS_TERRITORY	ISO_LANGUAGE	ISO_TERRITORY	NLS_CODESET	ISO_LANGUAGE_3
ar	AR	8	ARABIC	UNITED ARAB EMIRATES	ar	AE	AR8ISO8859P6	ara
bg	BG	101	BULGARIAN	BULGARIA	bg	BG	CL8ISO8859P5	bul
ca	CA	102	CATALAN	CATALONIA	ca	CT	WE8ISO8859P1	cat
cs	CS	30	CZECH	CZECH REPUBLIC	cs	CZ	EE8ISO8859P2	ces
de	D	4	GERMAN	GERMANY	de	DE	WE8ISO8859P1	deu
da	DK	5	DANISH	DENMARK	da	DK	WE8ISO8859P1	dan
es	E	11	SPANISH	SPAIN	es	ES	WE8ISO8859P1	spa
eg	EG	118	EGYPTIAN	EGYPT	eg	EG	AR8ISO8859P6	egy
el	EL	104	GREEK	GREECE	el	GR	EL8ISO8859P7	ell
es-US	ESA	29	LATIN AMERICAN SPANISH	AMERICA	es	US	WE8ISO8859P1	spa
fr	F	2	FRENCH	FRANCE	fr	FR	WE8ISO8859P1	fra
fr-CA	FRC	3	CANADIAN FRENCH	CANADA	fr	CA	WE8ISO8859P1	fra
en-GB	GB	1	ENGLISH	UNITED KINGDOM	en	GB	WE8ISO8859P1	eng
hr	HR	103	CROATIAN	CROATIA	hr	HR	EE8ISO8859P2	hrv
hu	HU	28	HUNGARIAN	HUNGARY	hu	HU	EE8ISO8859P2	hun
it	I	108	ITALIAN	ITALY	it	IT	WE8ISO8859P1	ita
is	IS	106	ICELANDIC	ICELAND	is	IS	WE8ISO8859P1	isl
he	IW	107	HEBREW	ISRAEL	he	IL	IW8ISO8859P8	heb
ja	JA	15	JAPANESE	JAPAN	ja	JP	JA16EUC	jpn
ko	KO	16	KOREAN	KOREA	ko	KR	KO16KSC5601	kor
lt	LT	109	LITHUANIAN	LITHUANIA	lt	LT	NEE8ISO8859P4	lit
no	N	10	NORWEGIAN	NORWAY	no	NO	WE8ISO8859P1	nor
nl	NL	6	DUTCH	THE NETHERLANDS	nl	NL	WE8ISO8859P1	nld
pl	PL	110	POLISH	POLAND	pl	PL	EE8ISO8859P2	pol
pt	PT	18	PORTUGUESE	PORTUGAL	pt	PT	WE8ISO8859P1	por
pt-BR	PTB	26	BRAZILIAN PORTUGUESE	BRAZIL	t	BR	WE8ISO8859P1	por
ro	RO	111	ROMANIAN	ROMANIA	ro	RO	EE8ISO8859P2	ron
ru	RU	112	RUSSIAN	RUSSIA	ru	RU	CL8ISO8859P5	rus
sv	S	13	SWEDISH	SWEDEN	sv	SE	WE8ISO8859P1	swe
fi	SF	7	FINNISH	FINLAND	fi	FI	WE8ISO8859P1	fin

Table 20–2 (Cont.) Language and Territory Values

LANGUAGE_TAG	LANGUAGE_CODE	LANGUAGE_ID	NLS_LANGUAGE	NLS_TERRITORY	ISO_LANGUAGE	ISO_TERRITORY	NLS_CODESET	ISO_LANGUAGE_3
sk	SK	113	SLOVAK	SLOVAKIA	sk	SK	EE8ISO8859P2	slk
sl	SL	114	SLOVENIAN	SLOVENIA	sl	SI	EE8ISO8859P2	slv
th	TH	115	THAI	THAILAND	th	TH	TH8TISASCII	tha
tr	TR	116	TURKISH	TURKEY	tr	TR	WE8ISO8859P9	tur
en	US	0	AMERICAN	AMERICA	en	US	US7ASCII	eng
zh-CN	ZHS	14	SIMPLIFIED CHINESE	CHINA	zh	CN	ZHS16CGB231280	zho
zh-TW	ZHT	117	TRADITIONAL CHINESE	TAIWAN	zh	TW	ZHT16BIG5	zho
sq	SQ	67	ALBANIAN	ALBANIA	sq	AL	EE8ISO8859P2	sqi
vi	VN	43	VIETNAMESE	VIETNAM	vi	VN	VN8MSWIN1258	vie
id	IN	46	INDONESIAN	INDONESIA	id	ID	WE8ISO8859P1	ind

20.6.2 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 20–3 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 20–3 Localization Database Attributes

Database Attribute	Default Value	Alter Session	Comments
`NLS_CALENDAR`	None	Never	This attribute need not be set as the default value is `GREGORIAN`. Accept the default value.
`NLS_COMP`	None	Never	This attribute need not be set as the default value is `BINARY`. Accept the default value.
`NLS_CURRENCY`	None	Never	Accept the default value.
`NLS_DATE_FORMAT`	`YYYY-MM-DD`	Create	Fixed value.
`NLS_DATE_LANGUAGE`	`NUMERIC DATE LANGUAGE`	Create	Fixed value. Does not require an attribute or profile.
`NLS_ISO_CURRENCY`	None	Never	Accept the default value.
`NLS_LANGUAGE`	None	Update	The value of this attribute is based on the `LANGUAGE` session attribute. See the `LANGUAGE` attribute in Table 20–1.
`NLS_TERRITORY`	`AMERICA`	Create	The value of this attribute is fixed. See the `TERRITORY` attribute in Table 20–1.
`NLS_LENGTH_SEMANTICS`	`CHAR`	Create	Fixed value. Does not require an attribute or profile.
`NLS_NCHAR_CONV_EXCP`	None	Never	Accept the default value. This parameter is not used.
`NLS_NUMERIC_ CHARACTERS`	`,.`	Create	Fixed value. Choose group and decimal separators independently.
`NLS_SORT`	None	Update	In order 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 20–1.
`NLS_TIME_FORMAT`	`HH24:MI:SS.FF`	Create	Fixed value. Does not require an attribute or profile.
`NLS_TIME_TZ_FORMAT`	`HH24:MI:SS.FF TZR`	Create	Fixed value. Does not require an attribute or profile.
`NLS_TIMESTAMP_FORMAT`	`YYYY-MM-DD HH24:MI:SS.FF`	Create	Fixed value. Does not require an attribute or profile.
`NLS_TIMESTAMP_TZ_ FORMAT`	`YYYY-MM-DD HH24:MI:SS.FF TZR`	Create	Fixed value. Does not require an attribute or profile.
`NLS_DUAL_CURRENCY`	None	Never	Accept the default value.

Note: As the language attributes tracked on the session reflect Java values, you cannot use them for formatting on the PL/SQL layer.

20.7 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.

Part IV

Developing Applications with Flexfields

This part of the Developer's Guide discusses how to use descriptive, extensible, and key flexfields to develop Oracle Applications that can be customized by application implementers and administrators without programming.

Getting Started with Flexfields introduces flexfield concepts and features, including the development process, development roles, and how flexfields appear in the user interface.

Using Descriptive Flexfields discusses the descriptive flexfield concepts and features, the development process, how to incorporate descriptive flexfields in user interface (UI) tables, forms, and query panels, and how to utilize descriptive flexfields with Oracle Business Intelligence, Web Services, and ADF Desktop Integration.

Using Extensible Flexfields discusses the extensible flexfield concepts and features, the development process, contexts, dedicated tables and views, how to employ extensible flexfields on an application page, how to programmatically accessing business component information, and customizing the runtime modeler.

Using Key Flexfields discusses the key flexfield concepts and features, the development process, producer and consumer development activities, how to incorporate key flexfields in UI tables and forms, and how to define and access key flexfield combination filters.

Testing and Deploying Flexfields discusses how to test your flexfield business components using the Integrated WebLogic Server, how to deploy your flexfield application to a standalone WebLogic Server to in order to test the full lifecycle, how to regenerate flexfield business components programmatically, and how to make flexfield setup task flows accessible from Oracle Fusion Functional Setup Manager.

This part contains the following chapters:

21

Getting Started with Flexfields

This chapter discusses the basics of using flexfields to enable customers to add custom attributes to business objects in their Oracle Fusion applications.

This chapter includes the following sections:

21.1 Introduction to Flexfields

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 development.

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 an "expandable" data field that is divided into *segments*. 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.

Customers 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*. The concepts of structure and context are defined later in this section. For information about flexfield configuration, see the "Using Flexfields for Custom Attributes" chapter in the *Oracle Fusion Applications Extensibility Guide*.

There are three types of flexfields, all of which enable implementers to configure application features without programming, and these configurations are fully supported within Oracle Fusion Applications:

- Descriptive — 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. Descriptive flexfields give customers the ability to extend the data model with additional attributes. A descriptive flexfield provides a set amount of segments for an entity, 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. For more information, see Section 21.1.1, "Descriptive Flexfields."

- 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 so they will always be presented together in the application user interface.
  - Hierarchical categories enable entities to inherit the segments that are configured for their parents.
  - Extensible flexfields support one-to-many relationships between the entity and the extended attribute rows.

  For more information, see Section 21.1.2, "Extensible Flexfields."
- Key — It can be difficult to anticipate what type of coding scheme or naming scheme customers might need to identify their business entities. Key flexfields give implementers and administrators the ability to configure their business entities to be identified using flexible, multi-part, "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 can be implemented to represent part numbers and account numbers. Key flexfields are never optional; customers must configure them for the product to operate correctly.

  For more information, see Section 21.1.3, "Key Flexfields."

To better understand flexfields and the differences between 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 rearrange the order of segments in a flexfield, you produce a different structure. In some applications, different 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.

  Your flexfield can display different segments and prompts for different end users based on a data condition in your application 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.
- Contexts — Descriptive and extensible flexfield segments are *context-sensitive* — they are varied or made available in your application by implementers and administrators to reflect the needs of the organization 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 is comprised of a set of context-sensitive segments that store 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 implementer can define a Dimensions context which might consist of segments that represent height, width and depth. The implementer can also define a Measurements context that contains segments that reuse the same underlying height, width and depth columns, and which also includes segments for weight, volume and density.

Note: Segments that are always available, regardless of context, are often referred to as *global* segments.

21.1.1 Descriptive Flexfields

Descriptive flexfields provide a way for customers to add custom attributes to entities, to define how the attributes are validated, and display properties for the attributes. These attributes are generally standalone; they don't necessarily have anything to do with each other and are not treated together as a combination. The segments of a descriptive flexfield that are made available to end users are exposed in the UI as individual fields.

Descriptive flexfields are entirely optional; customers can choose to configure and expose them or not, as they wish. For example, one company could configure the parts flexfield to store depth, height, and width, and another company could configure the parts flexfield to store size and color. Some companies might not need any additional attributes.

You create a descriptive flexfield for one of two purposes:

- For use by customer implementers and administrators — a *customer* flexfield.

  In this case you define and register the descriptive flexfield in your application database, but all configuration is accomplished by the application implementer or administrator, 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, then create contexts and segments, and define 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 implementers and administrators to extend the flexfield at your discretion, allowing them to configure it like a customer flexfield.

Even if administrators never change a flexfield once 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.

21.1.2 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.

Another feature of extensible flexfields is that you can have more than one context associated with any particular row of data, and a row can have multiple occurrences of the same context. That is, extensible flexfields support a one-to-many relationship between the entity and its extended attribute rows.

Extensible flexfields also enable implementers 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* for can be defined for extensible flexfields, and implementers 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 product 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.

21.1.3 Key Flexfields

Key flexfields are configurable multi-part "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 can be implemented to represent part numbers and account numbers.

A key flexfield is implemented in the UI as a collection of fields that can be displayed in various UI formats. Key flexfields are never optional; customers must configure them for the product to operate correctly.

The ability to individually tailor key flexfield structures to the 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. General Ledger determines which flexfield structure to use based on the value of a **Set of Books** user profile option.

Oracle has configured this flexfield to include six segments: company code, cost center, account, product, product line, and sub-account. Oracle has also defined valid values for each segment, as well as cross validation rules to describe valid segment combinations. However, other companies might structure their general ledger account fields differently. By including the Accounting Flexfield, General Ledger can accommodate the needs of different organizations. One company can configure the Accounting Flexfield structure to include six segments, while another company includes twelve segments, all without programming.

21.1.4 Value Sets

An end user enters a value into a flexfield segment while using your application. The flexfield validates each segment against a set of valid values — a *value set* — which is usually predefined by the application implementer or administrator. To "validate a segment" means that the flexfield compares the value that the user enters into the segment against the values that comprise the value set which 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, 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.

Value set metadata is mined and used by the business component modeler to create expert mode 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 information about value set configuration, see the "Using Flexfields for Custom Attributes" chapter in the *Oracle Fusion Applications Extensibility Guide*. For more information about the taxonomy hierarchy, see Appendix A, "Working with the Application Taxonomy".

21.1.5 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.

Flexfields can be included in Oracle Business Intelligence. However, because the polymorphic view objects used to model flexfields are not compatible with Oracle Business Intelligence, the flexfields 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 flexfields and incorporate them into your application, and when the application implementer or administrator configures the flexfields.

For more information, see Section 22.12, "Preparing Descriptive Flexfield Business Components for Oracle Business Intelligence" and Section 23.4.3, "How to Prepare Key Flexfield Business Components for Oracle Business Intelligence".

21.2 Participant Roles

Responsibility for flexfield development activities is allocated between *owners* and *implementers*. These are not formal development roles; they are used only in this documentation to clarify and group the flexfield development activities.

Owner

The flexfield 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 segment values or combination IDs (CCID) on a transaction table, and works with the structural and seed data and the business components that have been configured by the flexfield producer.

Implementer

A flexfield implementer is an individual who sets up all or part of a flexfield-enabled application for deployment. Implementers 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 implementer.

21.3 The Flexfield Development Life Cycle

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 in a seed database.
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 your application's business components.
4. Incorporate the flexfield into application pages.
5. Define sample flexfield data and use it to test the flexfield.
6. Use the tester mode of the flexfield business components wizard to create a model that you can use to test the flexfield.
7. Optionally, share your flexfield business components with other developers using an ADF library.

Once you have completed the flexfield development process and delivered your application, implementers can use the Manage Flexfields task flows 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. For information about planning and implementing flexfield configuration, such as defining structures, contexts, attributes, labels, behavior, and associated value sets, see the "Using Flexfields for Custom Attributes" chapter in the *Oracle Fusion Applications Extensibility Guide*.

Note: You must use the specified tools throughout the development lifecycle to create and update the flexfield metadata and application 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.

21.4 Flexfields in the Application User Interface

Flexfields consist of label/widget pairs or table fields. Once configured, extensible flexfields appear in the UI as regions, and 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 widget (text field, choice list, and so on) that allows a 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 the developer places 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.

22

Using Descriptive Flexfields

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:

22.1 Introduction to Descriptive Flexfields

Descriptive flexfields provide a way for customers to add custom attributes to entities, and to define validation and display properties for them. A descriptive flexfield is a logical grouping of attributes (segments) that are mapped to a set of extension columns, which are shipped as part of Oracle Fusion Applications tables. The attributes in the group are of two types: *global segments* and *context-sensitive segments*. 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*. To learn more about global, context, and context-sensitive segments, see Chapter 21, "Getting Started with Flexfields."

You can define multiple *usages* for a descriptive flexfield. For example, you might have defined an address flexfield that the implementer may use to add attributes related to addresses. The implementer 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 customer only needs to configure the flexfield 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.

   See Section 22.2, "Developing Descriptive Flexfields".

2. Create descriptive flexfield business components.

   See Section 22.3, "Creating 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.

   For more information, see Section 25.4, "Regenerating Flexfield Business Components Programmatically".

3. Create view links between the descriptive flexfield business components and your application's business components.

   See Section 22.4, "Creating Descriptive Flexfield View Links".

4. Nest the descriptive flexfield application module instance in the application module.

   See Section 22.5, "Nesting the Descriptive Flexfield Application Module Instance in the Application Module".

5. Add a descriptive flexfield view object instance to the application module.

   See Section 22.6, "Adding a Descriptive Flexfield View Object to the Application Module".

6. Add the descriptive flexfield usage to the appropriate application pages.

   See Section 22.7, "Adding Descriptive Flexfield UI Components to a Page".

7. Configure the descriptive flexfield UI components.

   See Section 22.8, "Configuring Descriptive Flexfield UI Components".

8. Load any necessary application seed data, such as error messages or lookup values.

   See Section 22.9, "Loading Seed Data."

After implementing a flexfield, you can define seed or test value sets for the flexfield, and you can create a model that you can use to test it. For more information, see Section 25.1.2, "How to Test Flexfields."

Once you have completed the flexfield development process and delivered your application, implementers 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. For information about planning and implementing flexfield configuration, such as defining attributes, labels, behavior, and

associated value sets, see the "Using Flexfields for Custom Attributes" chapter in the *Oracle Fusion Applications Extensibility Guide*.

To make the Manage Descriptive Flexfields task flows available to application implementers, you register them with the Oracle Fusion Functional Setup Manager. For more information, see Section 25.5, "Integrating Flexfield Task Flows into Oracle Fusion Functional Setup Manager".

22.1.1 The Benefits of Descriptive Flexfields

Descriptive flexfields let you satisfy different groups of users without having to reprogram your application, by enabling you to provide customizable fields for non-key attributes. Descriptive flexfields also enable context-sensitive fields that appear only when needed. In essence, a descriptive flexfield enables customers to extend the data model without writing either XML or Java. A 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 customers form would normally 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, since these are attributes of the customer entity 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 keep track of. You can add a descriptive flexfield to the 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 users enter on other parts of the page.

You could use a descriptive flexfield in a fixed assets application that you build to allow further description of a fixed asset. You could let the structure of the assets flexfield depend on the value of an asset type field. For example, a company could configure the flexfield 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."

Oracle General Ledger includes a descriptive flexfield in its journal entry form that implementers can configure to allow users to add information of their own choosing. For example, end users might need 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 customer implementers, consider defining a descriptive flexfield for every entity in your application to which a customer might need to add attributes.

22.1.2 How Descriptive Flexfields are Modeled in Oracle Application Development Framework

Flexfields are modeled as a collection of Oracle Application Development Framework (ADF) polymorphic view rows. 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 should be used. Given a collection of polymorphic view rows, each row can be a different type.

The attribute sets associated with the discriminator are predefined. In fact, the Oracle Application Development Framework enables each view row to have its own view

definition. When a polymorphic collection is created, the framework selects a view definition for the row to be added based on the value of the discriminator attribute.

With flexfields, this behavior is exposed using the terminology of segments and structures in place of attributes and view row types, respectively. The attributes in each view row definition are exposed as a set of segments with a predefined structure (or segment inclusion and ordering). The structure can include the discriminator attribute, common attributes that are unaffected by the discriminator value, and variable attributes that are included based on the discriminator value.

For descriptive flexfields, the context segment is the discriminator attribute, and the global segments are the common attributes.

Descriptive flexfields use a context switching mechanism similar to that of polymorphic view objects. You use a wizard to generate a flexfield polymorphic view object based on the descriptive flexfield definition, then create a view link to connect your product view object and the flexfield view object. You can then use the view object to add the flexfield to an application page.

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 be defaulted using a constant value from an underlying entity object, it is initialized to that value.
- If a segment is defined to be defaulted by derivation using a descriptive flexfield parameter, 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 *Oracle Fusion Middleware Fusion Developer's Guide for 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 Section 22.14, "Accessing Descriptive Flexfields from an ADF Desktop Integration Excel Workbook" and the *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.

22.2 Developing Descriptive Flexfields

Whenever you have an application table that you think customers might need to extend for their specific circumstances, you can add additional columns to the table and register those columns as flexfield segments. Once you have registered a flexfield, you have the ability to reuse the flexfield with other application tables.

To complete the process for developing a descriptive flexfield:

1. Add extension columns to the application table.

2. Register the flexfield and define its metadata and base table usage. You can register the flexfield using a registration task or using procedures from the FND_FLEX_DF_SETUP_APIS PL/SQL package.
3. Optionally reuse the flexfield by adding the same set of extension columns to other application tables.
4. Register all reuses using either a registration task or procedures from the FND_FLEX_DF_SETUP_APIS PL/SQL package.
5. Register the entity details for each usage.
6. Register flexfield parameters to identify external reference data sources for setting defaults and validating value sets. You can register the parameters using a registration task or using the procedures from the FND_FLEX_DF_SETUP_APIS PL/SQL package.

22.2.1 How to Create Descriptive Flexfield Columns

To implement a descriptive flexfield for an application table, you first add extension columns to that table. You need to add a context column, such as ATTRIBUTE_CATEGORY, and as many generic attribute (segment) columns of each type, such as ATTRIBUTE1_VARCHAR2 and ATTRIBUTE12_NUMBER, that you think the customers will need. A segment column must be a VARCHAR2, NUMBER, DATE, or TIMESTAMP. When defining a flexfield attribute, the implementer will need to map the 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 ATTRIBUTEn_VARCHAR2, ATTRIBUTEn_NUMBER, ATTRIBUTEn_DATE, and ATTRIBUTEn_TIMESTAMP. 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 implementation-defined context codes.

Each implementer can configure as many of the segment columns as the customer requires and can choose whether to use the context column.

You must use the Database Schema Deployment Framework tools to create the application table and columns. Using these tools ensures that the table and its columns are registered in the Applications Core Data Dictionary. For more information, see Chapter 56, "Using the Database Schema Deployment Framework."

22.2.2 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.
- Define the base table usage.
- Define the base table column to be used for the context segment.
- Map the base table columns to the flexfield segments.

You can define a descriptive flexfield using a registration task, which you access from the Oracle Fusion Applications Setup and Maintenance work area, or using the `FND_FLEX_DF_SETUP_APIS` PL/SQL package.

22.2.2.1 Registering and Defining Descriptive Flexfields Using a Registration Task

You can use the Register Descriptive Flexfields task, which is accessed from the Oracle Fusion Applications Setup and Maintenance work area, to register and define a descriptive flexfield. First you add a descriptive flexfield entry, then you define the base table usage. This becomes the master usage for the flexfield. You next specify which base table column to use for the context segment, and you define the base table columns to be used for the flexfield segments. You can then specify the name of the entity object, the package name, and the prefix to use when generating the flexfield's business components, or you can complete that task at a later time.

> **Tip:** You can also define and register a flexfield using the `FND_FLEX_DF_SETUP_APIS` PL/SQL package, as described in Section 22.2.2.2, "Registering and Defining Descriptive Flexfields Using the Setup APIs.".

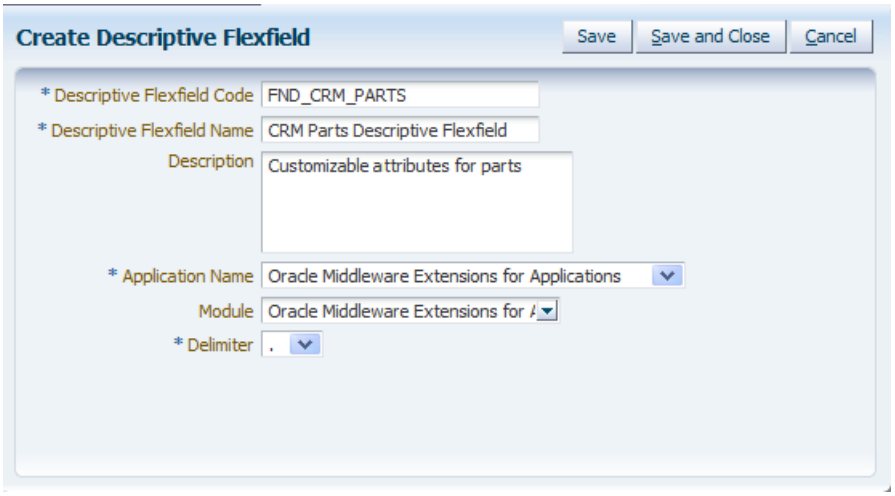
Before you begin:

Create the extension columns as described in Section 22.2.1, "How to Create Descriptive Flexfield Columns."

To register and define a descriptive flexfield using a registration task:

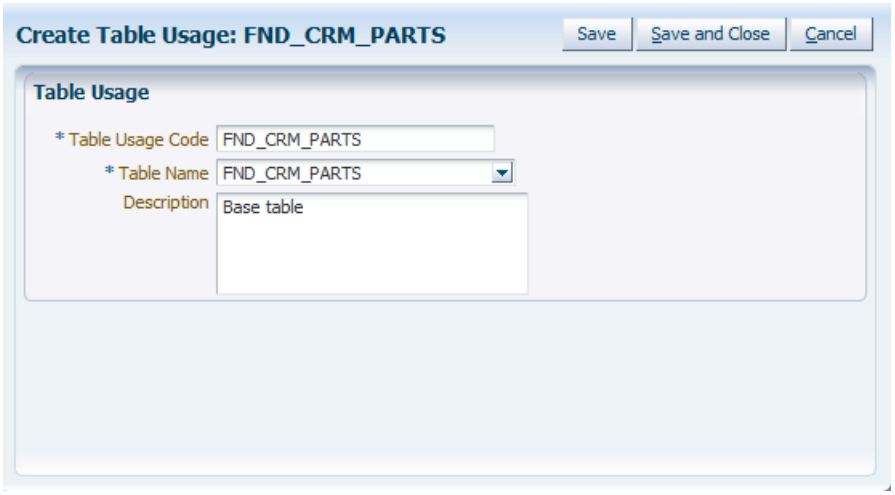
1. In the Oracle Fusion Applications global area, choose **Setup and Maintenance** from the **Administration** menu.
2. Go to the Register Descriptive Flexfields task.
3. From the Search Results section, select **Actions > New**.
4. On the Create Descriptive Flexfield page shown in Figure 22–1, set the following values:
   - **Descriptive Flexfield Code**: Provide a code that uniquely identifies the flexfield.
   - **Descriptive Flexfield Name**: Provide a descriptive name for the flexfield.
   - **Description**: Provide a short description of the flexfield.
   - **Application Name**: Select the application name that has been assigned for the product area.
   - **Module**: Select the seed data module.
   - **Delimiter**: Select the character to be displayed between flexfield segments, when the segments are displayed in a concatenated format.

Figure 22–1 Create Descriptive Flexfield Page



5. Click **Save and Close**.
6. In the Search Results section, select the flexfield that you just created and click **Primary Table**.
7. In the Descriptive Flexfield Table section of the Manage Flexfield Primary Table and Columns page, choose **Actions > New**.
8. On the Create Table Usage page shown in Figure 22–2, set the following values to define the base table usage for the flexfield:
   - **Table Usage Code**: Provide a code that uniquely identifies the flexfield usage. For the base table usage, this is typically the same code as the flexfield code.
   - **Table Name**: Type or select the name of the database table that contains the columns to be used as flexfield segments.
   - **Description**: Provide a short description of the flexfield usage.

Figure 22–2 Create Table Usage Page



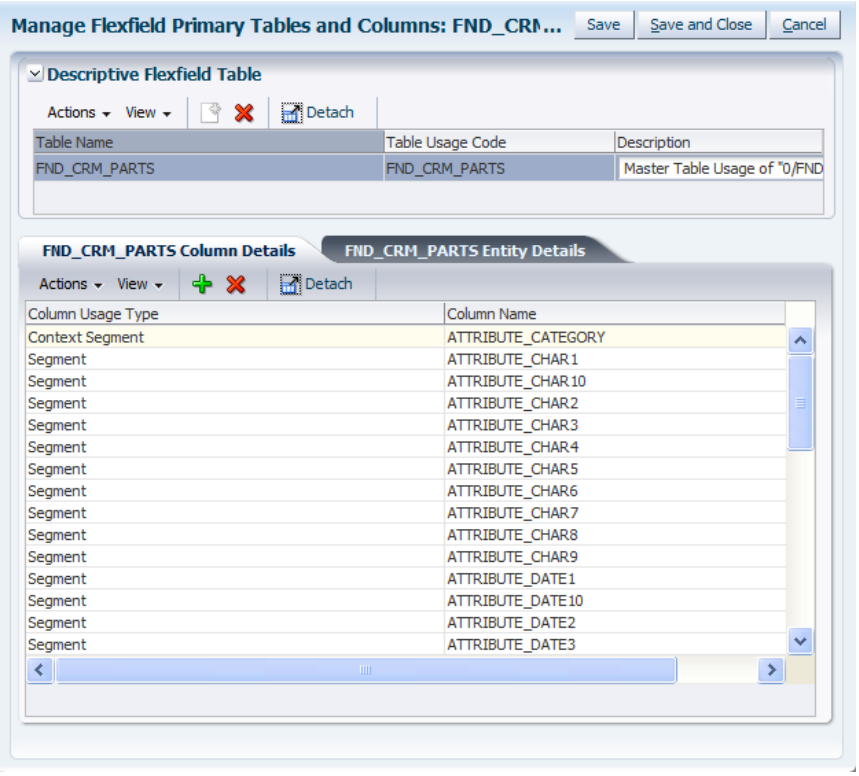
9. Click **Save and Close**.
10. In the *Table Name* **Column Details** tab, choose **Actions > New**.

11. Select **Context Segment** from the **Column Usage Type** dropdown list.
12. From the **Column Name** 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.

13. Repeat the following steps for each table column that you want to map to a flexfield segment, as shown in Figure 22–3.
    a. In the **Column Details** tab, choose **Actions > New**.
    b. Select **Segment** from the **Column Usage Type** dropdown list.
    c. Select the database column from the **Column Name** dropdown list that you want to map to a segment of the flexfield.

Figure 22–3 Column Details Tab



14. Click **Save and Close**.
15. You can register the entity details now or later. However, this step must be completed before you can generate the flexfield usage's business components. For more information, see Section 22.2.5, "How to Register Entity Details."
16. (Optionally) Register secondary usages of the flexfield as described in Section 22.2.4, "How to Register the Reuse of a Descriptive Flexfield."

22.2.2.2 Registering and Defining Descriptive Flexfields Using the Setup APIs

In addition to using a registration task, as described in Section 22.2.2.1, "Registering and Defining Descriptive Flexfields Using a Registration Task," 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 about flexfield definitions.

To learn how to access documentation about using the FND_FLEX_DF_SETUP_APIS PL/SQL package, see Section 22.2.2.2.1, "What You May Need to Know about the Descriptive Flexfield Setup API."

Before you begin:

Create the extension columns as described in Section 22.2.1, "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 master usage.
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 Section 22.2.5, "How to Register Entity Details." This step must be completed before you can generate the flexfield usage's business components.

22.2.2.2.1 What You May Need to Know about the Descriptive Flexfield Setup API

In the descriptive flexfield development process, you 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 outputs package documentation and usage examples to 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
```

22.2.3 How to Reuse a Descriptive Flexfield on Another Table

A descriptive flexfield configuration can be shared with other application tables. To reuse a descriptive flexfield, you add the same set of extension columns to the application table for which you want to reuse the flexfield, and you register the reuse as described in Section 22.2.4, "How to Register the Reuse of a Descriptive Flexfield."

The application table that was used to first register the flexfield is referred to as the *master* application table, and it is the *owner* of the flexfield. 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 master table. The secondary extension columns must have the same data type and size as the corresponding master table extension columns.

The column names must also be exactly the same as in the master usage, with the exception of an optional prefix. For example, if the column names are ATTRIBUTE1 and ATTRIBUTE2 in the master usage, 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 and HOME_ATTR2, or any variation that does not end in the names of the master usage columns.

You must use the Database Schema Deployment Framework tools when you create the application table and columns. Using these tools ensures that the table is registered in FND_TABLES, and its columns are registered in FND_COLUMNS.

22.2.4 How to Register the Reuse of a Descriptive Flexfield

After you add extension columns to a table for a reuse of a flexfield, you must register the usage before you can build a descriptive business component for the secondary usage.

You can register the reuse of a descriptive flexfield using a registration task or using procedures from the FND_FLEX_DF_SETUP_APIS PL/SQL package.

22.2.4.1 Registering the Reuse of a Descriptive Flexfield Using a Registration Task

You can use the Register Descriptive Flexfields task, which is accessed from the Oracle Fusion Applications Setup and Maintenance work area, to register the reuse of a descriptive flexfield by a secondary table. The registration task uses the master usage column mappings to determine how to map the secondary table's column names to the flexfield segments. If you specify a prefix, the task uses the prefix to determine the column mappings. For example, if the master table's ATTRIBUTE1 column is mapped to a segment, and you specify a prefix of HOME for the secondary usage, the task automatically maps its HOME_ATTRIBUTE1 column to a flexfield segment.

> **Tip:** You can also register the reuse of a flexfield using the FND_FLEX_DF_SETUP_APIS PL/SQL package, as described in Section 22.2.4.2, "Registering the Reuse of a Descriptive Flexfield Using the Setup APIs.".

Before you begin:

1. Register the descriptive flexfield and its master usage as described in Section 22.2.2, "How to Register and Define Descriptive Flexfields."
2. Create the extension columns for the reuse as described in Section 22.2.3, "How to Reuse a Descriptive Flexfield on Another Table."

To register a secondary reuse using a registration task:

1. In the Oracle Fusion Applications global area, choose **Setup and Maintenance** from the **Administration** menu.
2. Go to the Register Descriptive Flexfields task.
3. In the Search Results section select the flexfield for which you want to add a secondary usage and click **Secondary Tables**.
4. In the Descriptive Flexfield Table section of the Manage Flexfield Secondary Tables and Columns page, choose **Actions > New**.
5. On the Create Table Usage page, set the following values to define the secondary table usage for the flexfield:
   - **Table Usage Code**: Provide a code that uniquely identifies the flexfield usage.
   - **Table Name**: Select the name of the database table that contains the columns to be used as flexfield segments for this secondary usage.
   - **Column Name Prefix**: If you used a prefix for the flexfield column names, type the prefix.
   - **Description**: Provide a short description of the flexfield usage.

6. Click **Save and Close**.
7. You can register the entity details now or later. However, this step must be completed before you can generate the flexfield usage's business components. For more information, see Section 22.2.5, "How to Register Entity Details."

22.2.4.2 Registering the Reuse of a Descriptive Flexfield Using the Setup APIs

In addition to using the registration task, as described in Section 22.2.4.1, "Registering the Reuse of a Descriptive Flexfield Using a Registration Task," you can define and register the reuse of a descriptive flexfield using procedures from the `FND_FLEX_DF_SETUP_APIS` PL/SQL package.

To learn how to access documentation about using the `FND_FLEX_DF_SETUP_APIS` PL/SQL package, see Section 22.2.2.2.1, "What You May Need to Know about the Descriptive Flexfield Setup API."

The definition of a descriptive flexfield usage includes the following information:

- The table name
- The code for identifying the secondary usage
- The column name prefix. For example, if the master 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 master usage as described in Section 22.2.2, "How to Register and Define Descriptive Flexfields."
2. Create the extension columns for the reuse as described in Section 22.2.3, "How to Reuse a Descriptive Flexfield on Another Table."

To register a secondary reuse using the setup APIs:

1. To register the secondary reuse 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 Section 22.2.5, "How to Register Entity Details." This step must be completed before you can generate the flexfield usage's business components.

22.2.5 How to Register Entity Details

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 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 can have its own package name.

You can use a registration task or the `FND_FLEX_DF_SETUP_APIS` PL/SQL package to register this information for a flexfield usage.

22.2.5.1 Registering Entity Details Using a Registration Task

You can use the Register Descriptive Flexfields task, which is accessed from the Oracle Fusion Applications Setup and Maintenance work area, to register a flexfield usage's entity details.

> **Tip:** You can also register entity details using the `FND_FLEX_DF_SETUP_APIS` PL/SQL package, as described in Section 22.2.4.2, "Registering the Reuse of a Descriptive Flexfield Using the Setup APIs."

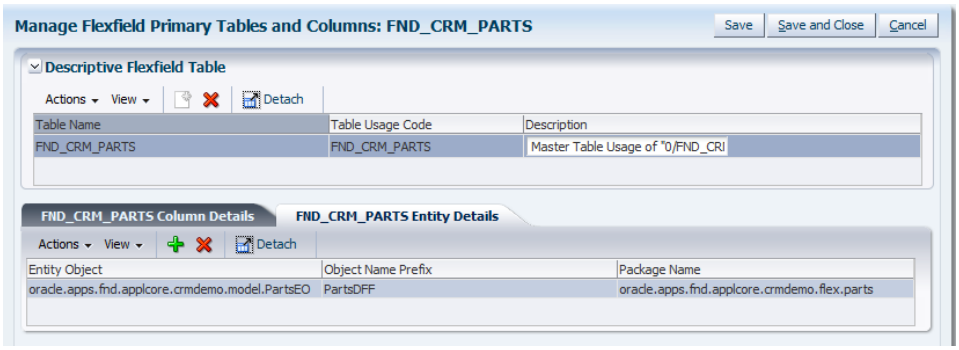
Before you begin:

1. Register the usage as described in Section 22.2.2, "How to Register and Define Descriptive Flexfields" and Section 22.2.4, "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 a registration task:

1. In the Oracle Fusion Applications global area, choose **Setup and Maintenance** from the **Administration** menu.
2. Go to the Register Descriptive Flexfields task.
3. In the Search Results section select the flexfield for which you want to add entity details and click **Primary Table** for the master usage or click **Secondary Tables** for a secondary usage.
4. Select the usage and click the *Table Name* **Entity Details** tab, as shown in Figure 22–4.

Figure 22–4 Entity Details Tab



5. If no row exists in the Entity Details section, choose **Actions > New** to create a row.
6. Set the following values:
   - **Entity Object**: Provide the full class name of the table's entity object.
   - **Object Name Prefix**: Type 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.

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.hcm.payroll.flex.dff1`, then you cannot define the `oracle.apps.hcm.payroll.flex` package for another usage, because that package would then identify both usages. Instead, you could use `oracle.apps.hcm.payroll.flex.dff2`.

7. Click **Save and Close**.

22.2.5.2 Registering Entity Details Using the Setup APIs

In addition to using the registration task, as described in Section 22.2.5.1, "Registering Entity Details Using a Registration Task," you can register entity details using procedures from the `FND_FLEX_DF_SETUP_APIS` PL/SQL package.

To learn how to access documentation about using the `FND_FLEX_DF_SETUP_APIS` PL/SQL package, see Section 22.2.2.2.1, "What You May Need to Know about the Descriptive Flexfield Setup API."

Before you begin:

1. Register the usage as described in Section 22.2.2, "How to Register and Define Descriptive Flexfields" and Section 22.2.4, "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.

22.2.6 How to Register Descriptive 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. Implementers 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 following ways:

- At row creation time to provide default segment values.
- In *derived* segments, whose values automatically change to reflect new reference values.
- In the WHERE clauses for table validated 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, application implementers are guaranteed that the parameters will always be mapped to entity object attributes, and they can use the parameters at will.

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 (non-flexfield) 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 implementer 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 Section 22.3, "Creating Descriptive Flexfield Business Components" and Section 22.8.3, "How to Configure Descriptive Flexfield Parameters".

22.2.6.1 Registering a Flexfield Parameter Using a Registration Task

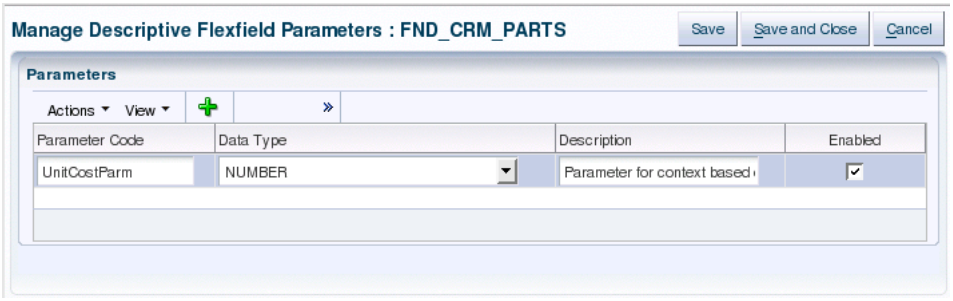
You can use the Register Descriptive Flexfields task, which is accessed from the Oracle Fusion Applications Setup and Maintenance work area, to register a flexfield parameter.

> **Tip:** You can also register parameters using the `FND_FLEX_DF_SETUP_APIS` PL/SQL package, as described in Section 22.2.6.1, "Registering a Flexfield Parameter Using a Registration Task."

To register a parameter using a registration task:

1. In the Oracle Fusion Applications global area, choose **Setup and Maintenance** from the **Administration** menu.
2. Go to the Register Descriptive Flexfields task.
3. In the Search Results section select the flexfield for which you want to add a parameter and click **Parameters**.
4. In the Parameters section of the Manage Flexfield Parameters page, choose **Actions > New**.
5. In the Parameters section shown in Figure 22–5, set the following values to define a parameter for the flexfield:
   - **Parameter Code**: Provide a code that uniquely identifies the parameter for the given flexfield.
   - **Description**: Provide a brief description of the parameter.
   - **Enabled**: Select this checkbox.

Figure 22–5 Manage Descriptive Flexfield Parameters Page



6. Click **Save and Close**.

22.2.6.2 Registering a Flexfield Parameter Using the Setup APIs

In addition to using the registration task, as described in Section 22.2.6.1, "Registering a Flexfield Parameter Using a Registration Task," you can register a flexfield parameter using procedures from the `FND_FLEX_DF_SETUP_APIS` PL/SQL package.

To learn how to access documentation about using the `FND_FLEX_DF_SETUP_APIS` PL/SQL package, see Section 22.2.2.2.1, "What You May Need to Know about the Descriptive Flexfield Setup API."

Before you begin:

Define and register the flexfield as described in Section 22.2.2, "How to Register and Define Descriptive Flexfields."

To create a parameter using the setup APIs:

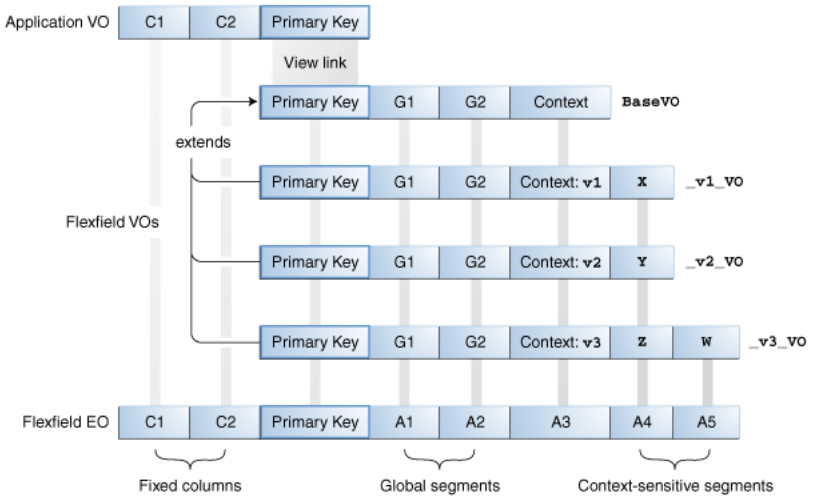
- Run the `fnd_flex_df_setup_apis.create_parameter(...)` procedure and provide a parameter code and data type.

22.3 Creating Descriptive Flexfield Business Components

Before you can use a descriptive flexfield in an application, you must generate flexfield Business Components for the flexfield. You generate these components by running a flexfield Business Component modeler. The modeler 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 application view object, and the *context* attribute, which serves as the discriminator.

When implementers configure the flexfields by defining global and context-sensitive segments, the base view object is regenerated and additional flexfield view objects are generated. Figure 22–6 shows an example of the configured components. The application view object contains only the non-flexfield 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 view object definition that represents the structure of the rows with that context value.

Figure 22–6 Descriptive Flexfield Modeled as ADF Business Components



Note: The application view object might contain other attributes. However, the application view object must not include flexfield view object attributes.

None of the flexfield view objects contains the fixed (non-flexfield) columns.

No Java implementation classes are generated for descriptive flexfield view objects. The application view object may or may not have Java implementation classes.

22.3.1 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.
- 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 *Oracle Fusion Applications Extensibility Guide*.
- 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, it must have a programmatically defaulted primary key.

- 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 Section 22.2.5, "How to Register Entity Details."

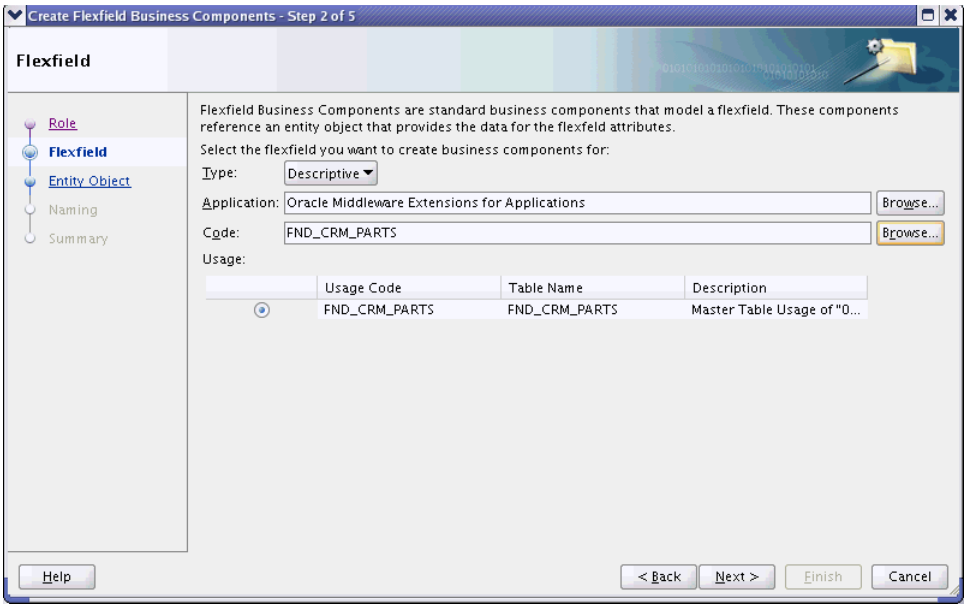
To create descriptive flexfield business components:

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, navigate to **Business Tier** > **ADF Business Components** and select **Flexfield Business Components**.
4. Click **OK**.
5. On the Role page of the Create Flexfield Business Components wizard, select the role that you are taking as you create the flexfield business components:
   - **`Developer`** — select this role if you are incorporating the flexfield into an application. The business components must be stored in one of your projects. Select the desired project location from the **Project Source Path** dropdown list.
   - **`Tester`** — select this role if you are planning to test your flexfield or a shared flexfield. In the **Output Directory** field, specify the path of your desired location for the generated business components.

     For more information about testing flexfields and importing shared flexfields, see Chapter 25, "Testing and Deploying Flexfields."

     Note: This is not a role in the security sense. It exists only during this procedure, for the purpose of specifying where your generated flexfield business components should be stored.
6. Click **Next**. The Flexfield page appears, as shown in Figure 22–7.

Figure 22–7 Create Flexfield Business Components Wizard — Flexfield Page

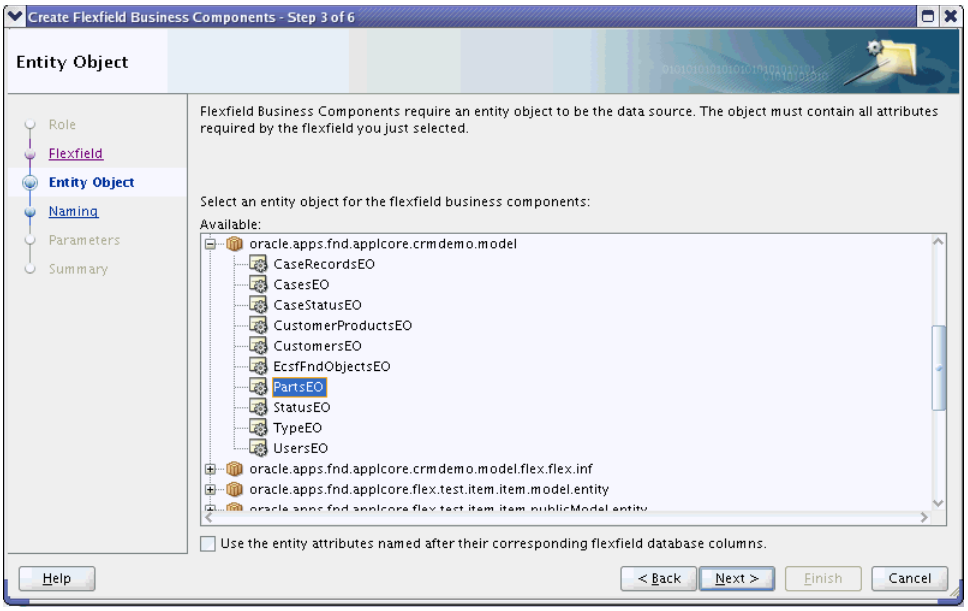


7. From the **Type** dropdown list, select `Descriptive`.
8. In the **Application** field, specify the full name of the application to which your descriptive flexfield belongs.

   You can browse for the name, and filter by **ID**, **Short Name**, or **Name**.
9. In the **Code** field, specify the code of the descriptive flexfield you want to use.

   You can browse for and filter by **Code**.
10. 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 master usage of the descriptive flexfield on the application table where it was originally defined. Every descriptive flexfield has one master usage.
    - A reuse (secondary usage) of the descriptive flexfield on an application table, including the one on which it was originally defined. Zero or more reuse instances can be defined for a given flexfield, each one potentially on a different application table. You can identify reuse instances by the presence of the prefix `(Reuse)` in the **Description** field.
11. Click **Next**.
12. On the Entity Object page, expand the tree of available models and select an entity object to use as the data source for the descriptive flexfield, as shown in Figure 22–8.

Figure 22–8 Create Flexfield Business Components Wizard — Entity Object Page



The entity object you select must include all of the attributes representing the columns that are reserved for the descriptive flexfield.

Descriptive flexfield attributes will be validated by the entity object along with its other attributes.

13. You might wish to select an entity object for which the descriptive flexfield attributes are defined as transient (not based on database table columns). If you need to do this, select the checkbox labeled **Use the entity attributes named after their corresponding flexfield database columns**. This checkbox is unselected by default.

    When a descriptive flexfield entity object attribute is transient, there is no matching underlying column name. When you select this checkbox, 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. Make sure 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 base table 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 Section 22.2.5, "How to Register Entity Details."

    Note: If the entity object with transient descriptive flexfield attributes is not based on the base table 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 Section 22.2.3, "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 upper case.

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 Section 22.2.5, "How to Register Entity Details." Review the names and click **Next**.

    If the selected entity object is not registered with the flexfield usage, the Naming page displays a message to that effect. Take one of the following actions:

    – Click **Back** to return to the Entity Object page and select an entity object that has been properly registered.

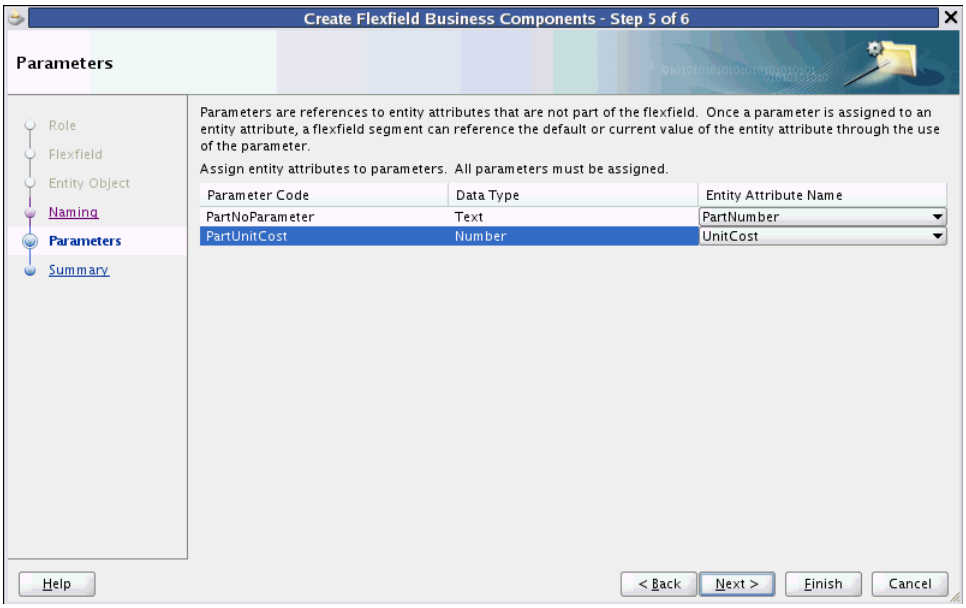
    – Click **Cancel** to exit this wizard and register the entity object that you want to use.

      For information about registering the entity object with the flexfield usage, see Section 22.2.5, "How to Register Entity Details."

16. On the Parameters page shown in Figure 22–9, 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 22–9 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 on 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 which is directly associated with the flexfield entity object through an accessor.
- Attributes that are part of any entity object which 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.

17. When all of the defined parameters are mapped, click **Next**.
18. On the Summary page, review your choices and 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 application project. You can resolve this issue by manually adding any missing libraries; then you can complete this procedure successfully.

19. 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.

22.4 Creating Descriptive Flexfield View Links

A view link is needed whenever an application view object references your descriptive flexfield. The application view object and the flexfield base view object are linked through their primary keys.

22.4.1 How to Create Descriptive Flexfield View Links

You create a view link to connect your product view object with the flexfield view object. Once you have created the view link, you can use the view object to add the flexfield to an application page.

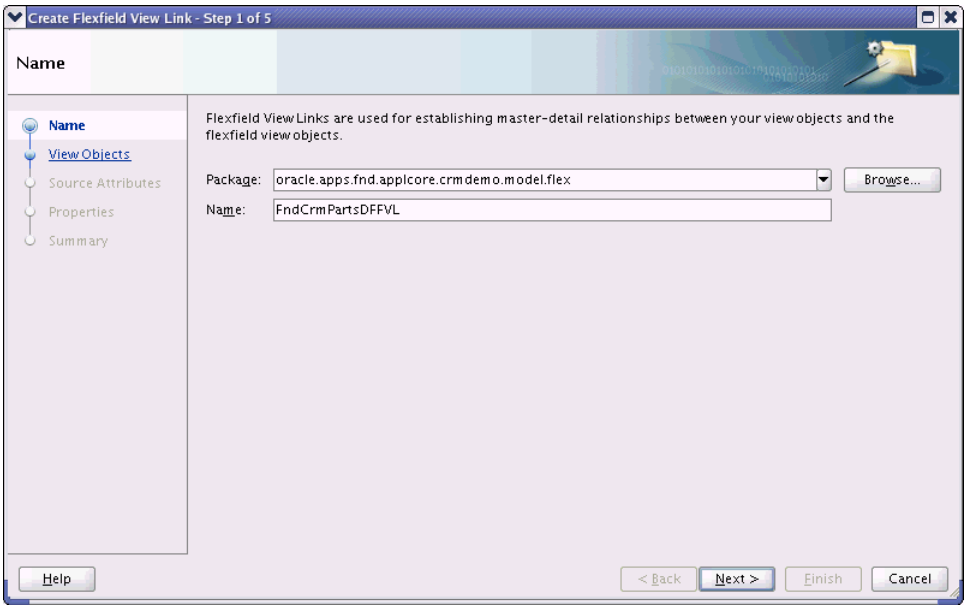
Before you begin:

1. Create the master view object, which contains only non-flexfield attributes.
2. Create the flexfield business components for the descriptive flexfield as described in Section 22.3.1, "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, navigate 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 22–10.

Figure 22–10 Create View Link Wizard — Name Page

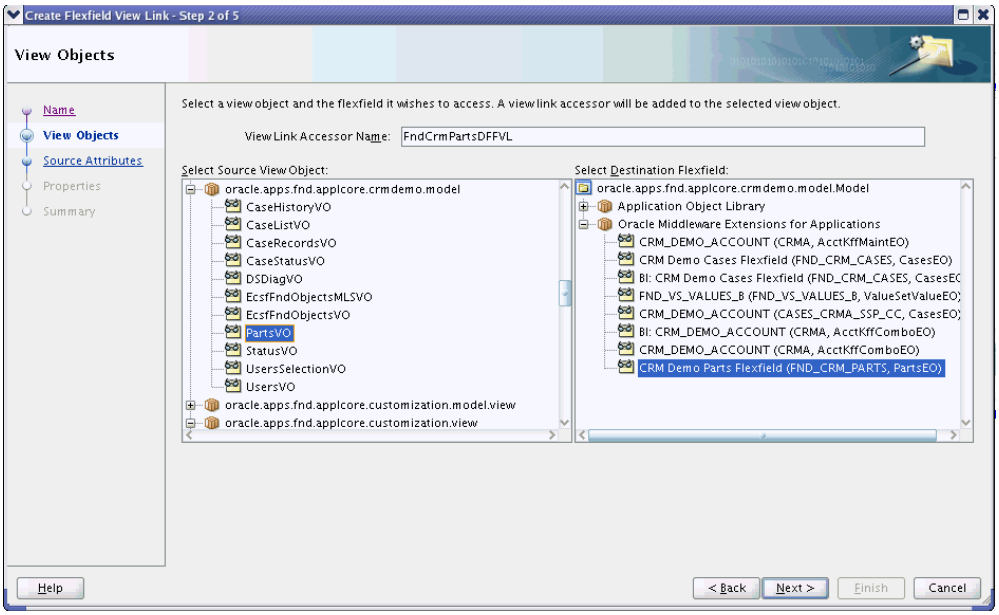


4. On the Name page, from the **Package** dropdown list, specify a package for the view link.

> **Caution:** You cannot move the view link to a different package after you create it. Instead, you must delete the view link and recreate 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 22–11.

Figure 22–11 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**.
11. Review the information on the Source Attributes page and click **Finish**.

    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.

12. On the Summary page, review the summary, then click **Finish**.

22.5 Nesting the Descriptive Flexfield Application Module Instance in the Application Module

You must nest the descriptive flexfield application module instance under the project application module before you can incorporate the descriptive flexfield usage into the UI.

You only need to nest one flexfield application module for a flexfield usage, even if two or more view links exist for the same flexfield usage.

22.5.1 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 nested descriptive flexfield application module instance shares the same transaction and entity object caches as the application module.

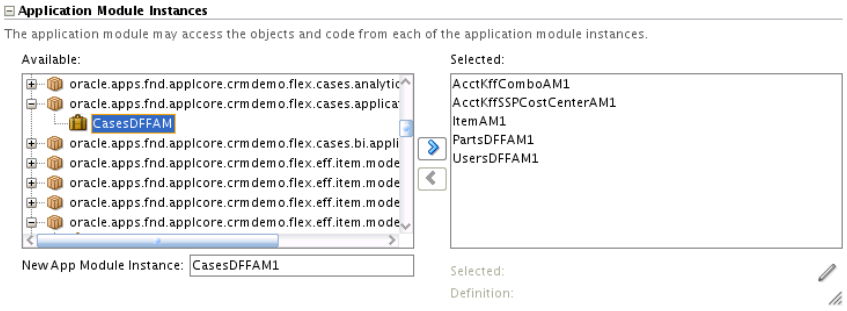
Before you begin:

1. You should have already created the project application module. For information about creating application modules, see the "Implementing Business Services with Application Modules" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
2. Create the master view object, which contains only non-flexfield attributes.
3. Create a flexfield business component for the descriptive flexfield usage as described in Section 22.3.1, "How to Create Descriptive Flexfield Business Components."

To nest the descriptive flexfield application module instance in the application module:

1. In the Application Navigator, double-click the project 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 22–12.

Figure 22–12 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 Section 22.2.5, "How to Register Entity Details."
5. Select the application module for the descriptive flexfield and shuttle 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 Section 22.2.5, "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.

22.6 Adding a Descriptive Flexfield View Object to the Application Module

You need to add a flexfield view object instance that reflects the hierarchy of the view link that you created in Section 22.4.1, "How to Create Descriptive Flexfield View Links" to the application module for your application.

22.6.1 How to Add a Descriptive Flexfield View Object Instance to the Application Module

You edit the project application module to add the flexfield view object.

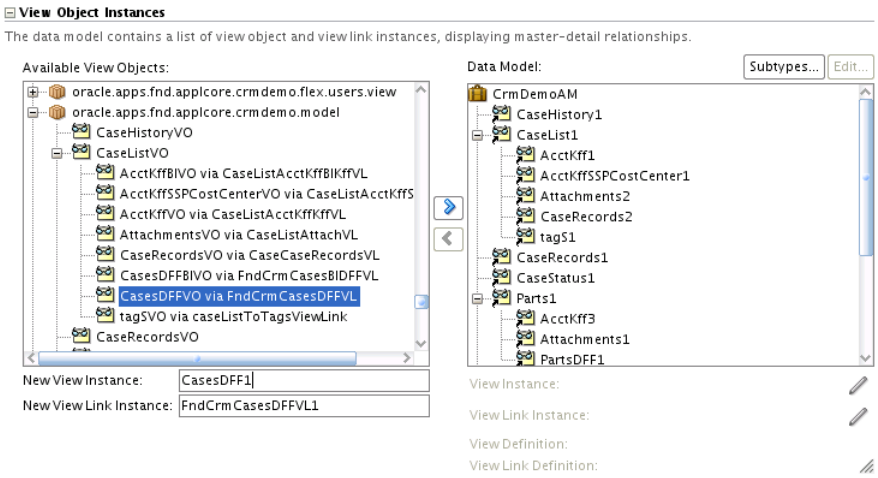
Before you begin:

1. You should have already created the project application module. For information about creating application modules, see the "Implementing Business Services with Application Modules" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
2. Create the master view object, which contains only non-flexfield attributes.
3. Create the flexfield business components for the descriptive flexfield usage as described in Section 22.3.1, "How to Create Descriptive Flexfield Business Components".
4. Create the flexfield view link as described in Section 22.4.1, "How to Create Descriptive Flexfield View Links."

To add a descriptive flexfield view object to the application module:

1. In the Application Navigator, double-click the project 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 22–13.

Figure 22–13 Application Module — View Object Instances Section



22.7 Adding Descriptive Flexfield UI Components to a Page

To include a descriptive flexfield on an application page, you 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 Section 22.11, "Incorporating Descriptive Flexfields Into a Search Form".

- Use ADF Desktop Integration to incorporate descriptive flexfields into an Excel workbook.

  See Section 22.14, "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 component is wrapped 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 method of adding flexfields to your application. The UI-first method is also available, but is not documented here.

22.7.1 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:

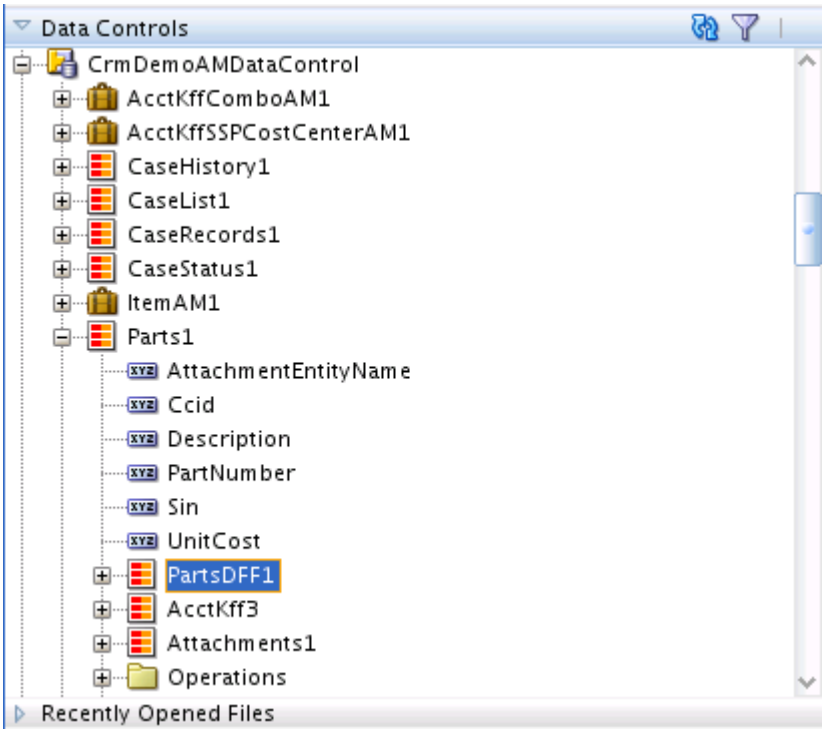
1. Create the descriptive flexfield view object as described in Section 22.3.1, "How to Create Descriptive Flexfield Business Components".
2. Create the view link as described in Section 22.4.1, "How to Create Descriptive Flexfield View Links".
3. Nest the descriptive flexfield application module instance in the project application module as described in Section 22.5.1, "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 Section 22.6.1, "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 22–14.

> **Caution:** You must use the flexfield view object child of the master view object. Do not use the flexfield view object from the flexfield's application module data control.

Figure 22–14 Flexfield View Object Nested Under Master View Object



4. Drag the flexfield view object onto a form, and select the appropriate flexfield UI component.

Tip: 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.

Tip: If a descriptive flexfield is in a region, such as a header, subheader, or tab, that does not contain core fields, there is a possibility that the customer will not use the flexfield segments and the region will be empty. To avoid the display of an empty region on the page, you should add controlling logic to hide the region if the customer has not defined the segments that appear in the region. For information about how to determine if a segment has been defined, see Section 22.10.2, "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 *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*

Tip: You can place the segments in a multiple-column layout, such as a two or three column layout. You should use a multiple-column layout when the number of segments that will be added by the customer 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 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 panelFormLayout component, with each row's content based on a different context value. For more information, see the "Arranging Content in Forms" section of the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

22.7.2 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. 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 Section 22.3.1, "How to Create Descriptive Flexfield Business Components".
2. Create the view link as described in Section 22.4.1, "How to Create Descriptive Flexfield View Links".
3. Nest the descriptive flexfield application module in the project application module as described in Section 22.5.1, "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 Section 22.6.1, "How to Add a Descriptive Flexfield View Object Instance to the Application Module".

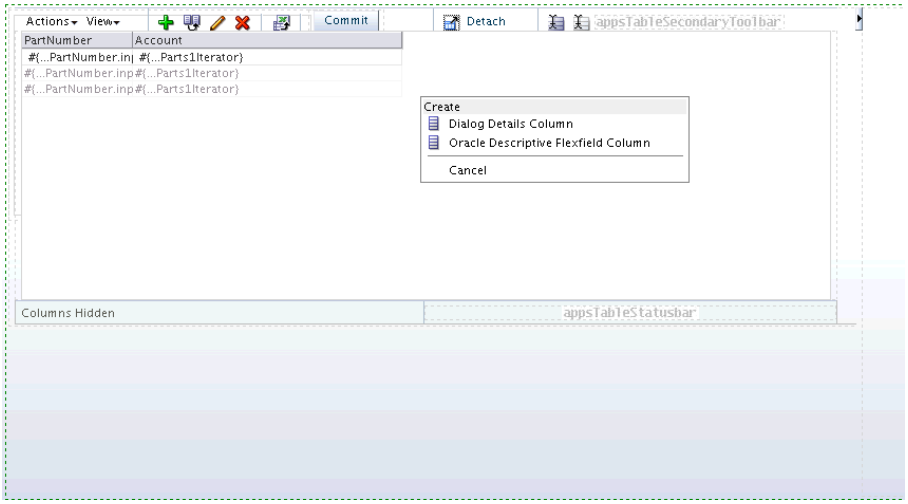
To add an unrestricted descriptive flexfield UI component to a table:

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**. Be sure to 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 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 22–15, and select **Oracle Descriptive Flexfield Column** as the UI component. This creates the base flexfield column in the table will the global segments will render.

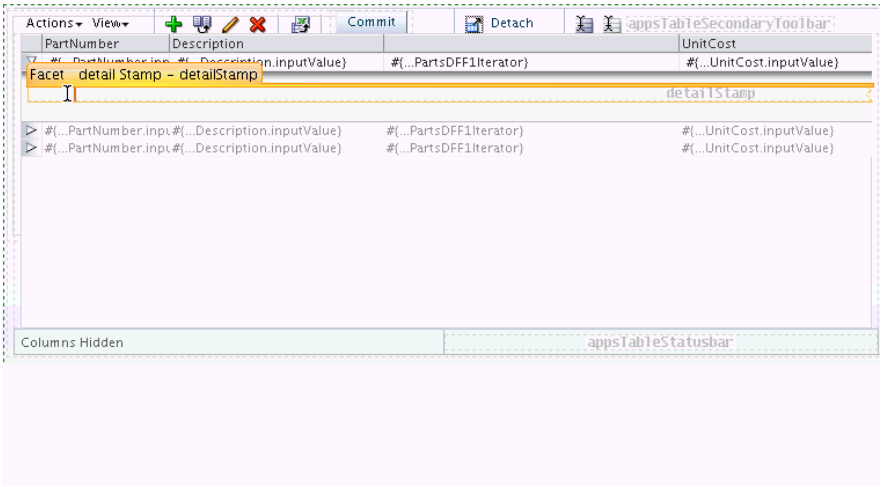
Figure 22–15 Descriptive Flexfield Dropped Into a Table



> **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.

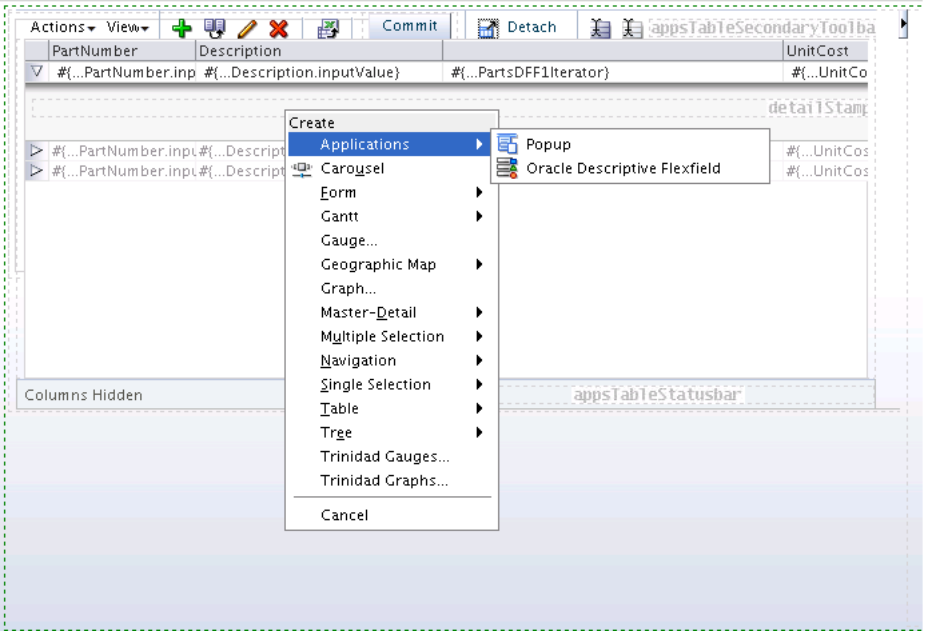
5. Create a detail region (`detailStamp` facet) if the table does not have one, as shown in Figure 22–16.

Figure 22–16 Detail Region — detailStamp Facet



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 facet on the Source tab or the structure view as shown in Figure 22–17; this creates the context sensitive fields.

Figure 22–17 Descriptive Flexfield Dropped Into a Detail Region



22.7.3 How to Add Descriptive Flexfield Context-Sensitive Segments to a Table as Columns

Normally, 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 appear 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 Section 22.3.1, "How to Create Descriptive Flexfield Business Components".
2. Create the view link as described in Section 22.4.1, "How to Create Descriptive Flexfield View Links".
3. Nest the descriptive flexfield application module in the project application module as described in Section 22.5.1, "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 Section 22.6.1, "How to Add a Descriptive Flexfield View Object Instance to the Application Module".

To add a descriptive flexfield UI component to a table as columns:

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**. Be sure to 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 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 22–15, 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 22–1 for information about the mode attribute.

6. Add an additional `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 value segment will be visible on the page, be sure to configure the segment to be read-only so end users cannot modify it. For more information, see Section 22.8.2, "How to Configure Segment-Level UI Properties".

22.7.4 How to Add Create Row and Delete Row Functionality to the Page

If your ADF Table component is wrapped in an Applications Table component, and if you are using your own `CreateInsert` 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 22–1 shows an example of the `pageDef` entry.

Example 22–1 Executables Element of Page Definition Code

```
<executables>
   <iterator Binds="Dff1RefInstance" RangeSize="25"
DataControl="Dff1RefAMDataControl" id="Dff1RefInstanceIterator"/>
   <iterator Binds="Dff1Instance" RangeSize="25"
DataControl="Dff1RefAMDataControl" id="Dff1InstanceIterator"/>
</executables>
```

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 22–1 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 22–2.

Example 22–2 Code to Set New Table Row State to STATUS_NEW

```
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);
}
```

6. Add code to the custom `createInsert` method to handle an empty table as described in Section 22.7.5, "How to Add a Row to an Empty Table in a Custom createInsert Method"

Caution: Please keep the following caveats in mind:

- 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 Section 22.8.1, "How to Configure Flexfield-Level UI Properties" and Section 22.8.2, "How to Configure Segment-Level UI Properties".

22.7.5 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 ApplicationTable component, you must include code similar to Example 22–3 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 22–3 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)
            {
              ((DescriptiveFlexfield) c1).createDynamicColumns();
            }
          }
        }
      }
    }
```

22.7.6 How to Dynamically Refresh a Descriptive Flexfield

If your flexfield is in an ADF Table component 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 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 22–4 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 22–5 shows the binding of the custom flexfield handler to the query component.

Example 22–4 Flexfield Listener

```
public void customDffSearchQueryListener(QueryEvent queryEvent)
{
  invokeMethodExpression(
    "#{bindings.DffCriteriaQuery.processQuery}",
    Object.class,QueryEvent.class,queryEvent);
  DescriptiveFlexfield.updateFlexColumns(appTable);
}
```

Example 22–5 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.customDffSearchQueryListener}"
  queryOperationListener="#{bindings.DffCriteriaQuery.processQueryOperation}"
  resultComponentId="::AT2:_ATp:ATt2"/>
```

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.

22.7.7 What Happens When You Add a Descriptive Flexfield to a Page

Descriptive flexfield segments appear on a form as a widget with a customer-defined label, just as core fields do. In tabular layout, the label of the flexfield segment is the column header and the values are within each cell of the column.

Figure 22–18 shows an example of a descriptive flexfield used in a form on an application page.

Figure 22–18 Example of a Descriptive Flexfield In a Form

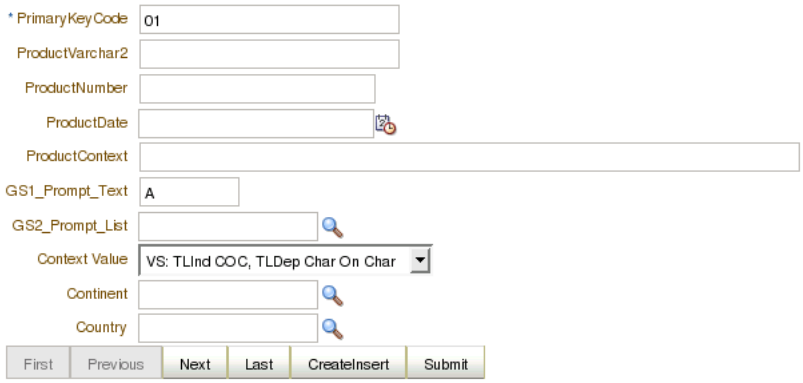
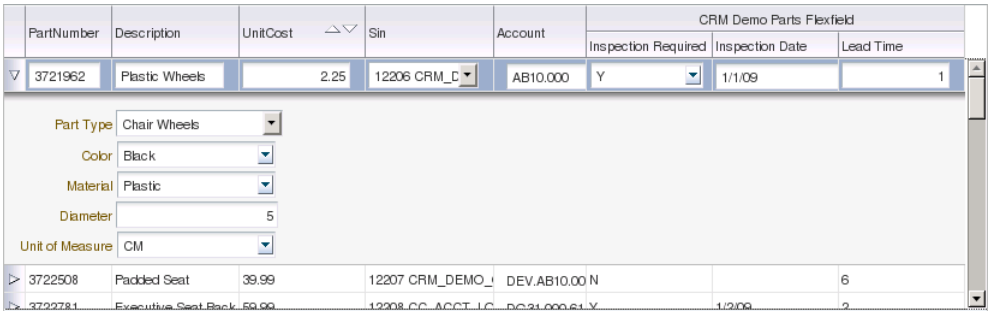


Figure 22–19 shows an example of a descriptive flexfield used in a table on an application page:

Figure 22–19 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.

22.8 Configuring 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.

22.8.1 How to Configure Flexfield-Level UI Properties

You configure flexfield-level behavior by configuring the UI component's properties.

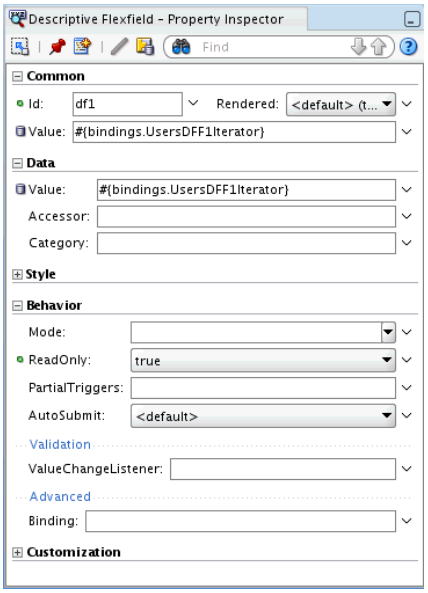
Before you begin:

Add the descriptive flexfield to the page as described in Section 22.7, "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 22–20.

Figure 22–20 Descriptive Flexfield Property Inspector



The significant properties on the Common, Data, Style and Behavior property tabs are listed in Table 22–1:

Table 22–1 Descriptive Flexfield Properties

Tab > Property	Description
Common > **Id**	The ID of the flexfield.
Common > **Rendered**	Indicates whether the flexfield is rendered on the page. Values can be `True` (default) or `False`. EL expressions are allowed.[1]
Common > **Value**	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.
Data > **Accessor**	The name of the accessor between the product team view object and the flexfield view object.
Data > **Category**	Defines which category will be rendered on the page. The category can be set on each attribute's custom property.
Style > **StyleClass**	The style class of the flexfield.
Style > **InlineStyle**	The inline style of the component.
Behavior > **Read-Only**	Indicates whether the flexfield is rendered as read-only. Values can be `True` or `False` (default). EL expressions are allowed.

Table 22–1 (Cont.) Descriptive Flexfield Properties

Tab > Property	Description
Behavior > **Mode**	Defines the UI mode of the descriptive flexfield component, to render all of the segments or just some of them. Values can be: ■ No value (default) — render all of the descriptive flexfield segments. ■ `global` — render only the global segments of the flexfield. This is the default value for a descriptive flexfield inside a table column, to generate sub-columns for the global segments. ■ `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.
Behavior > **partialTriggers**	The IDs of the components that should trigger a partial update in the flexfield (`String[]`). EL expressions are allowed.
Behavior > **valueChangeListener**	A method reference to a value change listener (`javax.faces.el.MethodBinding`). Requires an EL expression.
Behavior > **binding**	An EL reference that will store the component instance on a bean (`oracle.apps.fnd.applcore.flex.ui.DescriptiveFlexfield`). Requires an EL expression.

[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.

22.8.2 How to Configure Segment-Level UI Properties

Descriptive flexfields support finer control of segments in the UI through the following segment-level boolean properties:

- **`rendered`** — Indicates whether the segment is visible on the application page.
- **`required`** — Indicates whether the segment must have a value.
- **`readOnly`** — Indicates whether users can modify the segment.

The default values of these properties are derived from the flexfield metadata, but you can override them by inserting customization elements into the UI metadata. You can set these properties in the flexfield XML with literal values or EL expressions in most cases.

> **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 required in the page metadata.

For information about using EL expressions, see the "Creating ADF Data Binding EL Expressions" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Before you begin:

Add the descriptive flexfield to the page as described in Section 22.7, "Adding Descriptive Flexfield UI Components to a Page".

To configure segment-level UI properties:

- In the Applications page of the Component Palette, drag the following components onto the descriptive flexfield component:

Flexfield Context Segment Hints

Use to configure the descriptive flexfield context segment. This component must be inserted as a child of the **Descriptive Flexfield** component.

You can configure any combination of the **ReadOnly**, **Rendered**, and **Required** properties.

Flexfield Segment Hints

Use to configure global segments, or as a wrapper for the **Flexfield Segment Hint** component to configure individual segments. This component must be inserted as a child of the **Descriptive Flexfield** component. Apply the properties as follows:

- To configure all global segments, do not provide a value for the **ContextCode** property. Configure any combination of the **ReadOnly**, **Rendered**, and **Required** properties.

  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.

- To configure specific global segments, omit all properties and insert one or more **Flexfield Segment Hint** components as children.
- To configure all context-sensitive segments within a particular context, provide a value for the **ContextCode** property and configure any combination of the **ReadOnly**, **Rendered**, and **Required** properties.
- To configure specific context-sensitive segments within a particular context, provide a value for the **ContextCode** property and insert one or more **Flexfield Segment Hint** components as children.

  Note: This property does not affect the descriptive flexfield context segment. For context segment configuration, see the entry for **Flexfield Context Segment Hints**.

Flexfield Segment Hint

Use to configure individual global or context-sensitive segments. This element must be inserted as a child of the **Flexfield Segment Hints** component.

Specify the **SegmentCode**, which should identify a context-sensitive segment within a particular context if the parent element contains a **ContextCode** value, or a global segment if not.

You can configure any combination of the **Rendered**, **Required**, or **ReadOnly** properties. The **ReadOnly** property is generally set when there is a default value assigned to that segment, and you do not want users to choose a value other than the default. You only should set both **Required** and **ReadOnly** to **true** if the segment has a default value

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.

To determine the correct value for the **SegmentCode** property, examine the `FND_ACFF_SegmentName` attribute of the context-sensitive segment's `viewAttribute` element in your descriptive flexfield view object.

Example 22–6 shows various combinations of the segment-level display properties in the flexfield XML

Example 22–6 Segment-Level Display Properties in Descriptive Flexfield Metadata

```
<fnd:descriptiveFlexfield value="#{bindings.Dff1Iterator}">
    <!-- Customize the context segment. -->
    <fnd:flexfieldContextSegmentHint rendered="#{expr}"/>

    <!-- Customize all global segments.
         For a descriptive flexfield that was added as table columns,
         the rendered property must apply to the entire column.
      -->
    <fnd:flexfieldSegmentHints rendered="#{expr}"
                               required="#{expr}"
                               readOnly="#{expr}"/>

    <!-- Customize a specific global segment.
         For a descriptive flexfield that was added as table columns,
         the rendered property must apply to the entire column.
      -->
    <fnd:flexfieldSegmentHints>
        <fnd:flexfieldSegmentHint segmentCode="GlobalX"
                                  rendered="#{expr}"
                                  required="#{expr}"
                                  readOnly="#{expr}"/>
    </fnd:flexfieldSegmentHints>

    <!-- Customize all segments for a certain context.
         This does not affect the context segment.  The hint for the context
         segment is specified by flexfieldContextSegmentHint.
      -->
    <fnd:flexfieldSegmentHints contextCode="Context5" readOnly="#{expr}"/>

    <!-- Customize a specific context-sensitive segment. -->
    <fnd:flexfieldSegmentHints contextCode="Context3">
        <fnd:flexfieldSegmentHint segmentCode="Extra 1"
                                  rendered="#{expr}"
                                  required="#{expr}"
                                  readOnly="#{expr}"/>
    </fnd:flexfieldSegmentHints>

<fnd:descriptiveFlexfield/>
```

22.8.3 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, as described in Section 22.2.6, "How to Register Descriptive Flexfield Parameters".
2. Add the descriptive flexfield to the page as described in Section 22.7, "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 make sure 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 22–7 shows the source view of the `partialTriggers` attribute.

Example 22–7 Adding the UI Component ID to the partialTriggers List

```
<fnd:descriptiveFlexfield value="#{bindings.Dff1Iterator}"
partialTriggers=customerInputText>
```

22.9 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 Seed Data Loader.

After you complete the registration process described in Section 22.2.2, "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 implementer. In addition to the registration data, your flexfield seed data might include contexts, segments, and value sets that you have defined for your flexfield.

For information about extracting and loading seed data, see Chapter 55, "Initializing Oracle Fusion Application Data Using the Seed Data Loader".

22.10 Working 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

22.10.1 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 flexfield view row from the `ValueChangeEvent` instance.

```
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.

22.10.2 How to Determine Whether Descriptive Flexfield Segments Have Been Defined

Your application might find it useful to know if any global or context-sensitive segments exist in a descriptive flexfield's metadata before deciding whether to invoke 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.

22.10.3 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 ValueChangeEvent, and the listener is called.

Before you begin:

Add the descriptive flexfield to the page as described in Section 22.7, "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 22–8 is an example of a method that handles a ValueChangeEvent.

Example 22–8 Sample Listener Handler

```
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 `valueChangeListener` property (described in Table 22–1). In the Property Inspector, click the **Edit** button for `valueChangeListener`, and a wizard appears to help you select an existing event listener or create a new listener.

Example 22–9 is an example of the metadata as an EL expression that identifies the `dffChangeListener` listener from the Java method in Example 22–8.

Example 22–9 Sample EL Expression Identifying dffChangeListener

```
<fnd:descriptiveFlexfield value="#{bindings.PJCDFF1Iterator}"
     valueChangeListener=
     "#{managed_DFFHeaderTablePropHandler.dffChangeListener}"
     autoSubmit="true">
```

For more information about handling value change events, see the "Using Input Components and Defining Forms" chapter in the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

22.11 Incorporating Descriptive Flexfields 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.

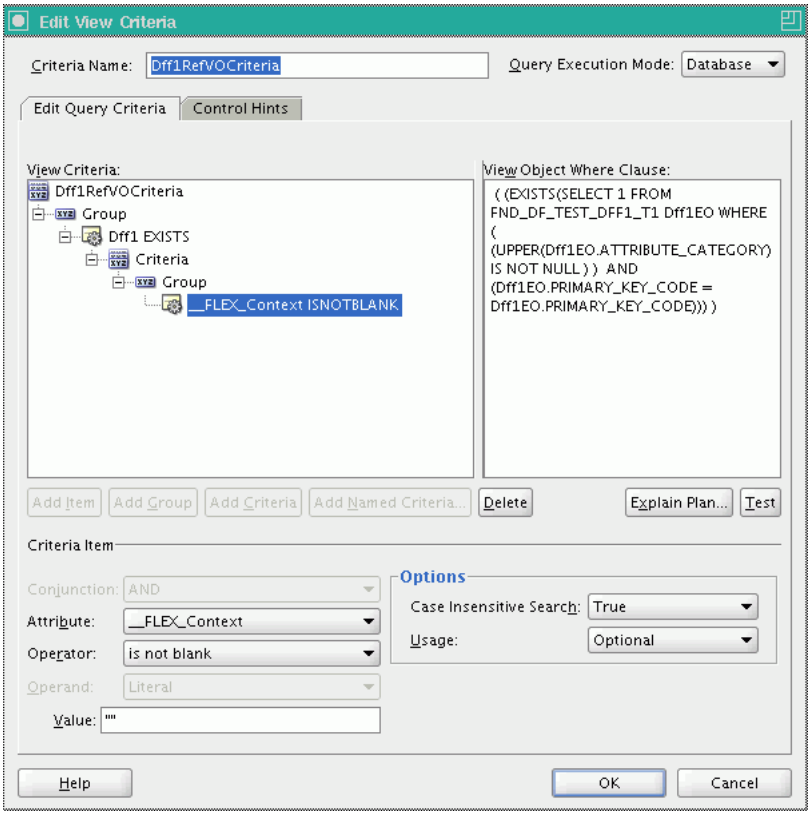
22.11.1 How to Incorporate Descriptive Flexfields Into a Search Form

You 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 a Query Panel with results to incorporate a descriptive flexfield into a search form.

To add a descriptive flexfield to a search form:

1. Add view criteria to the application view object instance that uses attributes from the view-linked descriptive flexfield view object, as shown in Figure 22–21.

Figure 22–21 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 discriminator and use it as the attribute in the view criteria definition.

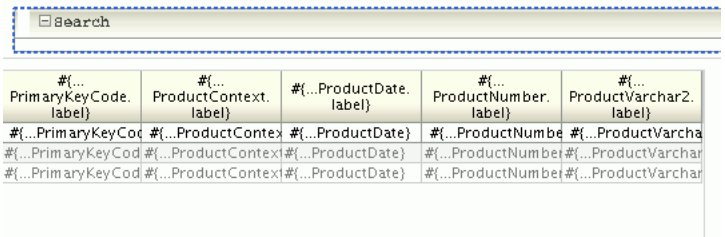
2. Create a new JSPX page in the user interface project.

> **Tip:** JDeveloper names the user interface project `ViewController` by default.

3. In the Data Control panel, select the named view criteria that you created previously.

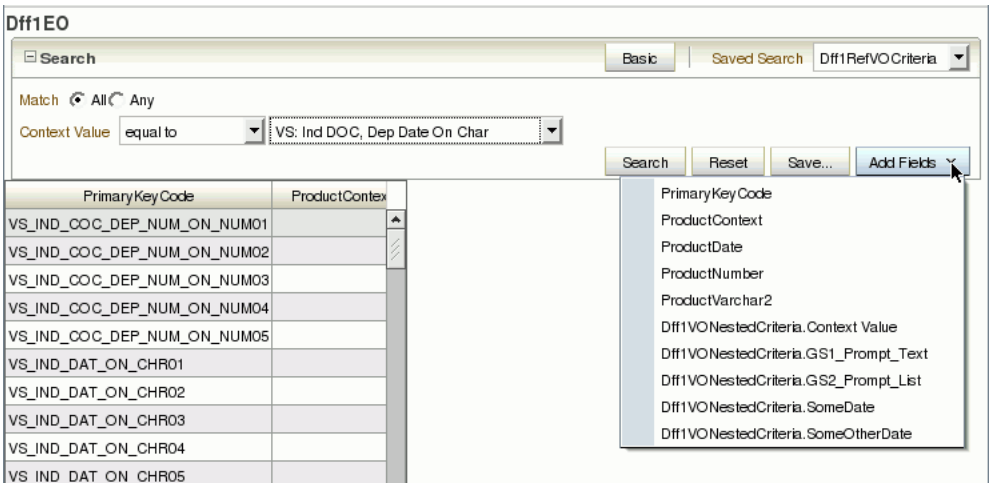
4. Drag and drop the view criteria onto the page as a Query Panel with results, as shown in Figure 22–22.

Figure 22–22 Page with Descriptive Flexfield Search Form



5. 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 22–23.

Figure 22–23 Descriptive Flexfield Search Form UI



For more information about working with search forms, see the "Creating ADF Databound Search Forms" chapter of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

22.12 Preparing 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.

While descriptive 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. To enable a descriptive flexfield to be used by Oracle Business Intelligence, it must be *flattened* into a usable static form. To flatten the flexfield into a static form you enable the flexfield and its segments for business intelligence (BI), you create the flexfield business components, and you create the flexfield view links and application modules using a slightly modified process.

22.12.1 How to Enable a Descriptive 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 enable the flexfield and its segments.

You can set the business intelligence–enabled flag at registration time using the `fnd_flex_df_setup_apis.create_flexfield(...)` procedure, or you can set the flag later using the `fnd_flex_df_setup_apis.update_flexfield(...)`

procedure. For information about using these procedures see Section 22.2.2.2.1, "What You May Need to Know about the Descriptive Flexfield Setup API."

You can enable the segments for business intelligence using the Manage Descriptive Flexfields task, which is accessed from the Oracle Fusion Applications Setup and Maintenance work area, as described in the "Configuring Descriptive Flexfields" section in *Oracle Fusion Applications Extensibility Guide*.

An alternative method to business intelligence–enable a descriptive flexfield and its segments is to set the `BIEnableFlag` to `Y` at both the descriptive flexfield level and the segment level in the descriptive flexfield seed data file (SDF), as shown in Example 22–10.

Example 22–10 BI-Enabled Descriptive Flexfield

```
<DescriptiveFlexfield>
    <ApplicationId>0</ApplicationId>
    <DescriptiveFlexfieldCode>FLEX_DFF1</DescriptiveFlexfieldCode>
    ...
    <Delimiter>.</Delimiter>
    <BIEnabledFlag>Y</BIEnabledFlag>
    ...
    <ContextSegment>
      <ContextCode>Context Data Element</ContextCode>
      <SegmentCode>Context Segment</SegmentCode>
      ...
      <ReadOnlyFlag>N</ReadOnlyFlag>
      <BIEnabledFlag>Y</BIEnabledFlag>
      ...
    </ContextSegment>
    ...
    <GlobalSegment>
       <ContextCode>Global Data Elements</ContextCode>
       <SegmentCode>GlobalSegment2</SegmentCode>
       ...
       <ReadOnlyFlag>N</ReadOnlyFlag>
       <BIEnabledFlag>Y</BIEnabledFlag>
       ...
    </GlobalSegment>
    ...
    <Context>
      <ContextCode>VS_FRM_CHR_ON_CHR</ContextCode>
      ...
      <Segment>
        <SegmentCode>L10</SegmentCode>
        ...
        <ReadOnlyFlag>N</ReadOnlyFlag>
        <BIEnabledFlag>Y</BIEnabledFlag>
        ...
      </Segment>
      ...
    </Context>
    ...
    <Context>
      <ContextCode>VS_FRM_CHR_ON_CHR2</ContextCode>
      ...
      <Segment>
        <SegmentCode>L10</SegmentCode>
        ...
        <ReadOnlyFlag>N</ReadOnlyFlag>
```

```
      <BIEnabledFlag>Y</BIEnabledFlag>
      <BIEqualizationTag>L10_TAG</BIEqualizationTag>
      ...
    </Segment>
    ...
  </Context>
  ...
<DescriptiveFlexfield>
```

22.12.2 How to Produce a Business Intelligence–Enabled Flattened Descriptive Flexfield Model

When you create business components for a business intelligence–enabled descriptive 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 descriptive flexfield polymorphic view object. You must also slightly modify the process of creating descriptive flexfield view links and application modules.

> **Note:** When you make changes to a business intelligence-enabled flexfield, you use the Import Metadata Wizard to import the changes into the Oracle Business Intelligence repository as described in the "Using Incremental Import to Propagate Flex Object Changes" section in the *Oracle Fusion Middleware Metadata Repository Builder's Guide for Oracle Business Intelligence Enterprise Edition (Oracle Fusion Applications Edition)*.

Before you begin:

1. Enable the flexfield and the desired segments for Oracle Business Intelligence as described in Section 22.12.1, "How to Enable a Descriptive Flexfield for Oracle Business Intelligence."
2. If your flexfield uses hierarchical value sets, you must make sure 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, see Section 59.8.1, "Designing a Column-Flattened View Object for Oracle Business Intelligence."

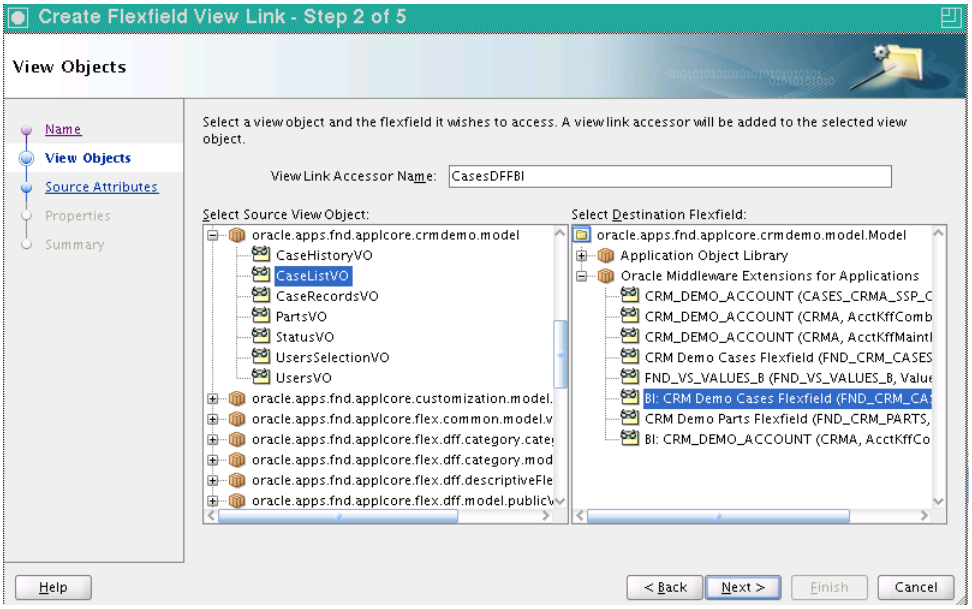
To produce a business intelligence–enabled flattened descriptive flexfield model:

1. Create descriptive flexfield business components as described in Section 22.3, "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 normal descriptive flexfield view object.
2. Create a view link using the procedure described in Section 22.4, "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 22–24.

Figure 22–24 Create Flexfield View Link Wizard — View Objects Page



3. Create an application module for use with Oracle Business Intelligence as described in Section 22.6, "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.

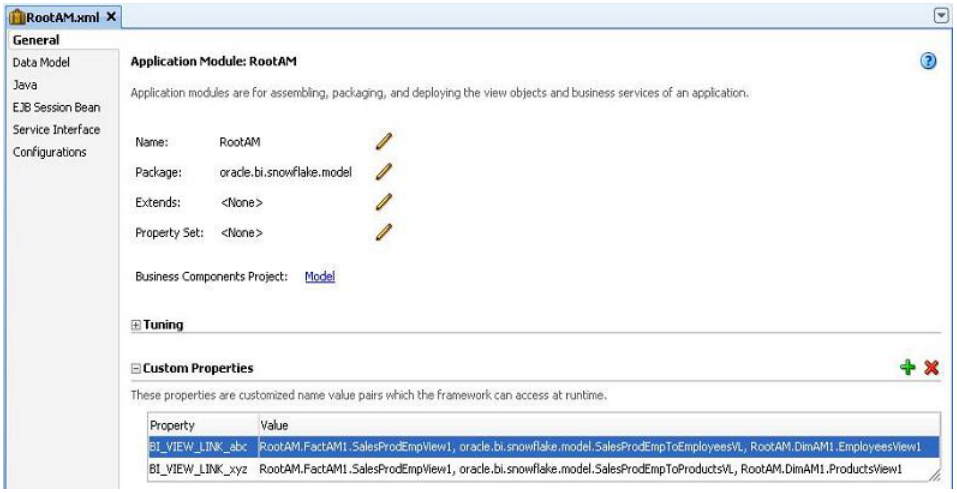
   b. On the Application Modules page of the wizard, add an instance of the descriptive flexfield Oracle BI application module as a nested instance of this application module. You can identify the Oracle BI application module by the **`analytics`** subpackage under the package root.

   Note: If you already have a product Oracle Business Intelligence application module, you may use it.

4. Following Oracle BI EE development guidelines, 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 22–25.

Figure 22–25 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 normally 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 you specify in different application modules are created during import from Oracle ADF if custom properties are defined on the application module.

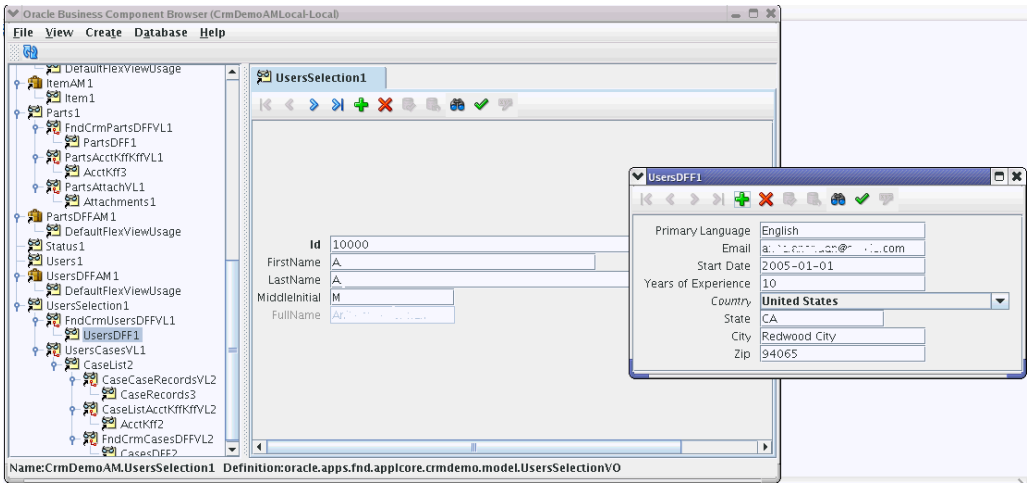
22.13 Publishing Descriptive Flexfields as Web Services

You can make access to a descriptive flexfield available through web services, which will enable you to perform CRUD (create, retrieve, update and delete) operations on the flexfield data rows. You accomplish this by exposing the descriptive flexfield application module as a web service and adding flexfield service data object support utility methods to the product application module.

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 22–6, 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 22–26 shows an application module tester view of a descriptive flexfield.

Figure 22–26 Application Module Tester 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.

22.13.1 How to Expose a Descriptive Flexfield as a Web Service

You make web service access to descriptive flexfields available by adding a custom property to the view link, service-enabling the master view object, exposing the application module as a web service, exposing operations on the master view object, and adding utility methods for the flexfield to the product application module. You can then deploy the service and run Java client programs to test the service as described in Section 22.13.2, "How to Test the Web Service."

For more information about service enabling an application module, see "Integrating Service-Enabled Application Modules" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*

Before you begin:

1. Create a flexfield business component for the flexfield's usage as described in Section 22.3.1, "How to Create Descriptive Flexfield Business Components."

> **Note:** When you generate a flexfield business component, the IDE automatically service enables the business component by generating a Service Data Object (SDO) for the base view object and for every subtype view object.

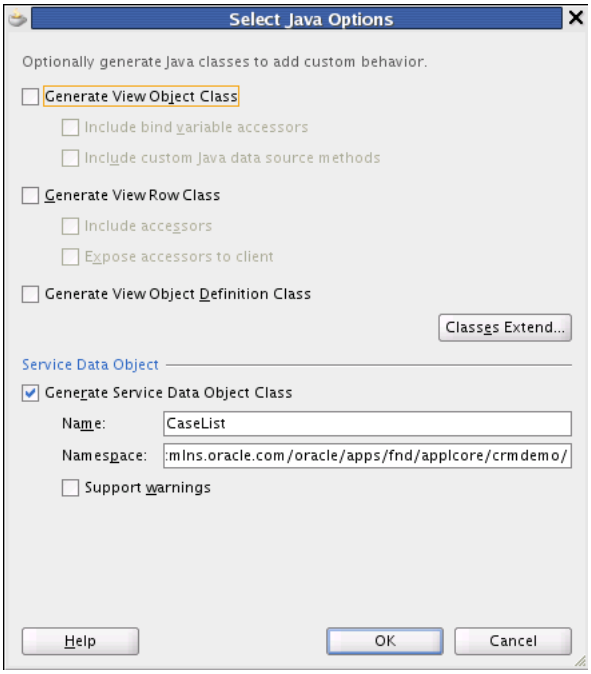
2. Create a flexfield view link between the master view object, which contains only non-flexfield attributes, and the flexfield business component for the flexfield usage as described in Section 22.4.1, "How to Create Descriptive Flexfield View Links."
3. Nest the descriptive flexfield application module instance in the project application module as described in Section 22.5.1, "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 Section 22.6.1, "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 base flexfield 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 non-flexfield 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** and click **OK**, as shown in Figure 22–27.

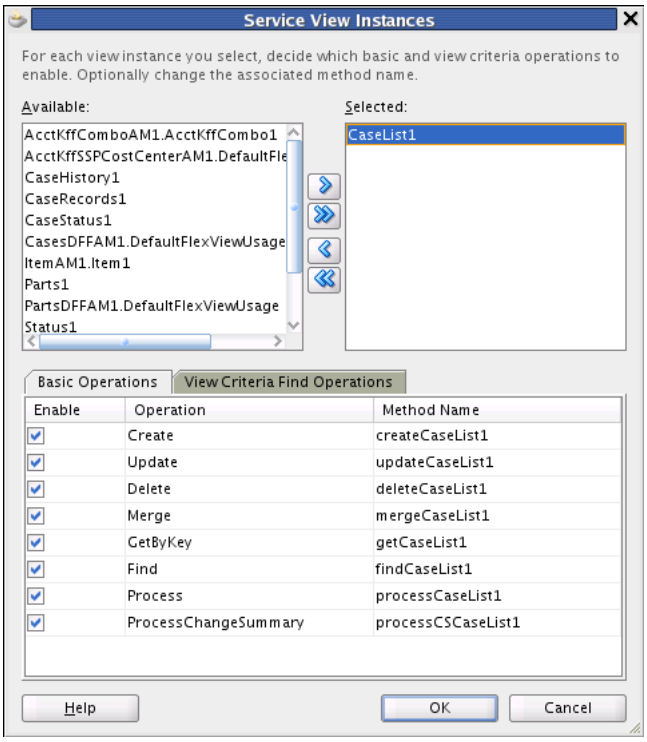
Figure 22–27 Generating a Service Data Object Class for the Master View Object



7. From the Application Navigator, open the product application module.
8. In the overview editor, click the **Service Interface** navigation tab.
9. 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** twice to go to the Service View Instances page.
   c. Shuttle 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 22–28, select all the operations in the **Basic Operations** list, and click **OK**.

Figure 22–28 Selecting Master View Object Instances to Expose



e. Click **Finish**.

10. 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. Shuttle 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 22–28, select all the operations in the **Basic Operations** list, and click **OK**.

11. 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`.

12. In the overview editor for the product application module, click the **Java** navigation tab and click the link for the **Application Module Class**.

13. Add the utility methods shown in Example 22–11 to the application module class. These utility methods enable web service clients to obtain `FlexfieldSdoSupport` objects to access flexfield information.

Replace *Flexfield* with the appropriate string for the flexfield that you are working with.

In the `getFlexfieldSdoSupport` method, replace *getDffAM* with the name of the getter method for the nested flexfield application module.

Example 22–11 Utility Methods for Flexfield Service Data Object Support

```
public List<String> getFlexfieldSdoNamespaceAndName(String contextValue)
{
  FlexfieldSdoSupport ss = getFlexfieldSdoSupport(contextValue);
  if (ss == null)
  {
    return null;
  }
  return Arrays.asList(ss.getSdoNamespace(), ss.getSdoName());
}

public String getFlexfieldTypeSdoPath()
{
  FlexfieldSdoSupport ss = getFlexfieldSdoSupport(null);
  if (ss == null)
  {
    return null;
  }
  return ss.getDiscriminatorSdoPath();
}

public List<String> getFlexfieldSegmentSdoPaths(String contextValue,
                                                List<String> segmentCodes)
{
  FlexfieldSdoSupport ss = getFlexfieldSdoSupport(contextValue);
  if (ss == null)
  {
    return null;
  }
  ArrayList r = new ArrayList(segmentCodes.size());
  for (String segmentCode: segmentCodes)
  {
    r.add(ss.getSegmentSdoPath(segmentCode));
  }
  return r;
}

public FlexfieldSdoSupport getFlexfieldSdoSupport(String contextValue)
{
  // Set contextValue to null to get the parent (base)
  // Find the nested flexfield application module instance
  DFFApplicationModuleImpl am =
   (DFFApplicationModuleImpl) getDffAM();
  return am.getSdoSupport(contextValue);
}
```

14. In the overview editor for the product application module, click the **Service Interface navigation** tab for the product application module and click the **Edit** icon in the **Service Interface Custom Methods** section.
15. In the Service Custom Methods dialog, shuttle 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.

22.13.2 How to Test the Web Service

You can test the service by adding `StringRefAddr` elements to the `Reference` element for the project 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 Section 22.13.1, "How to Expose a Descriptive Flexfield as a Web Service."
- Make sure that the BC4J Service Client and BC4J Service Runtime libraries are included in your application project.

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 project application module's service (DFF1MasterApplicationModuleService in this example).
3. Add the `StringRefAddr` elements that are shown in bold in Example 22–12 to the `Reference` element for the project application module's service. Modify the host and port number in the `jndiProviderURL` entry to point to your WebLogic Server. The port number is typically 7101.

Example 22–12 StringRefAddr Elements to Add to Service Reference in connections.xml

```
<Reference
name="{http://xmlns.oracle.com/oracle/apps/fnd/applcore/flex/test/dff1/model/}DFF1
MasterApplicationModuleService" className="oracle.jbo.client.svc.Service"
xmlns="">
   <Factory className="oracle.jbo.client.svc.ServiceFactory"/>
   <RefAddresses>
      <StringRefAddr addrType="serviceInterfaceName">

<Contents>oracle.apps.fnd.applcore.flex.test.dff1.model.DFF1MasterApplicationModul
eService</Contents>
      </StringRefAddr>
      <StringRefAddr addrType="serviceEndpointProvider">
         <Contents>ADFBC</Contents>
      </StringRefAddr>
      <StringRefAddr addrType="jndiName">

<Contents>DFF1MasterApplicationModuleServiceBean#oracle.apps.fnd.applcore.flex.tes
t.dff1.model.DFF1MasterApplicationModuleService</Contents>
      </StringRefAddr>
      <StringRefAddr addrType="serviceSchemaName">
         <Contents>DFF1MasterApplicationModuleService.xsd</Contents>
      </StringRefAddr>
      <StringRefAddr addrType="serviceSchemaLocation">
         <Contents>oracle/apps/fnd/applcore/flex/test/dff1/model/</Contents>
      </StringRefAddr>
      <StringRefAddr addrType="jndiFactoryInitial">
         <Contents>weblogic.jndi.WLInitialContextFactory</Contents>
      </StringRefAddr>
      <StringRefAddr addrType="jndiProviderURL">
         <Contents>t3://localhost:port_number</Contents>
      </StringRefAddr>
      <StringRefAddr addrType="jndiSecurityPrincipal">
         <Contents>weblogic</Contents>
      </StringRefAddr>
      <StringRefAddr addrType="jndiSecurityCredentials">
```

```
        <Contents>weblogic1</Contents>
      </StringRefAddr>
   </RefAddresses>
</Reference>
```

4. Run the remote server class for the product application module to deploy the service to the 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 Section 22.13.1, "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 22–13 shows how to get a data row. The XML payload output of this program is shown in Example 22–14.
   - Example 22–15 shows how to create a new data row.
   - Example 22–16 shows how to update an existing data row.

Example 22–13 Web Service Get Operation

```
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){
            e.printStackTrace();
        }
    }

    public static void main(String args[]){
          DFFTester dfftester =new DFFTester();
          dfftester.getFKRowGivenKey();
    }
}
```

Example 22–14 Example XML Payload Output of Web Service Get Operation

```
<?xml version="1.0" encoding="UTF-8"?>
<ns2:MasterDff
xmlns:ns2="http://xmlns.oracle.com/apps/fnd/applcore/flex/test/dff1/model/"
xmlns:ns1="http://xmlns.oracle.com/apps/fnd/applcore/flex/test/dff1/flex/dff1/"
xmlns:ns0="http://xmlns.oracle.com/adf/svc/types/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="ns2:MasterDff">
<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:dff1 xsi:type="ns1:Dff1VS_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>
</ns2:dff1>
</ns2:MasterDff>
```

Example 22–15 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 =
                dataFactory.create(dffInfo.get(0), dffInfo.get(1));
            System.out.println(dff1Service.getMyFlexfieldTypeSdoPath());
            dffSubType.set(dff1Service.getMyFlexfieldTypeSdoPath(), contextValue);
            dffSubType.set(dff1Service.getMyFlexfieldSegmentSdoPaths(contextValue,
              Arrays.asList("GLOBAL_STATE_ID_NUM")).get(0),
                           new BigDecimal(100));
            dffSubType.set(dff1Service.getMyFlexfieldSegmentSdoPaths(contextValue,
              Arrays.asList("Date_On_Date")).get(0),
                           java.sql.Date.valueOf("2011-04-18"));
            dffMaster.setDff1(dffSubType);
            dff1Service.createMasterDff1(dffMaster);


            MasterDffImpl masterVOSDO =
                (MasterDffImpl)dff1Service.getMasterDff1(PK1);
            String updatedXML =
                DFFTester.extractXMLFromDataObject(dff1Service, masterVOSDO);
            System.out.println("<<<--------Updated XML output-------------->");
            System.out.println("");
            System.out.println(updatedXML);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
...
```

Example 22–16 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.getMyFlexfieldTypeSdoPath(), contextValue);

            dffSubType.set(dff1Service.getMyFlexfieldSegmentSdoPaths(contextValue,
              Arrays.asList("GlobalSegment1")).get(0),
                           new BigDecimal(02));
            dffSubType.set(dff1Service.getMyFlexfieldSegmentSdoPaths(contextValue,
              Arrays.asList("P6_S0_On_Number")).get(0),
                           new BigDecimal(123));
            dffMaster.setDff1(dffSubType);
            dff1Service.updateMasterDff1(dffMaster);


            dffMaster = (MasterDffImpl)dff1Service.getMasterDff1(PK1);
            String updatedXML =
                DFFTester.extractXMLFromDataObject(dff1Service, dffMaster);
            System.out.println("<<<--------Updated XML output-------------->");
            System.out.println("");
            System.out.println(updatedXML);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
...
```

22.14 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 Desktop Integration Developer's Guide for Oracle Application Development Framework* 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.
- Incorporating a descriptive flexfield as a dynamic component or a single cell on the page.
- Creating an Excel workbook to integrate with the descriptive flexfield.
- Preparing your Excel workbook to access the column containing the flexfield.

The *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework* 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 Table component.

  A web page picker popup dialog can be associated with a dynamic column, enabling 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 Table component.

- 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 Table component, 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, 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 Form component, 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 and 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 user can control the context value, handle user-initiated context values changes in the dynamic column descriptive flexfield.
3. Handle the update or insert of a descriptive flexfield data row.

22.14.1 How to Configure ADF Desktop Integration with a Dynamic Column Descriptive Flexfield

When you configure the ADF Table component, 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 which 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 *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.

- To make the descriptive flexfield column of the Table component dynamic, set the **DynamicColumn** property in the TableColumn array of the ADF Table component 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 *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.

- 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.

22.14.2 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 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 user of the workbook, changes are not an issue. However, if the user can control the context value (for example, using an LOV in a "header" form), your application must be able to respond appropriately to update the descriptive flexfield structure.

After the user specifies a context value, you must invoke the worksheet `UpSync` method to get the new value into the model. Then you can use the ADF Table component `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 Table component `DownloadForInsert` method or the `Initialize` method to enable the Table component to reconfigure to accommodate the new flexfield structure.

22.14.3 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 Table component. If you are adding your descriptive flexfield as a static column, you can alternatively use an ADF Read-only Table component. Keep in mind that ADF Read-only Table components support static columns, but not dynamic columns. Popup dialogs support both types.

> **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. 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 (a descriptive flexfield global segment attribute) 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 descriptive flexfield.

  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 *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.

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 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.

22.14.4 How to Handle Update or Insert of a Descriptive Flexfield Data Row

To handle update or insert of a data row containing a descriptive flexfield in an ADF Desktop Integration table, you call a custom application module method which 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` 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, since this controls the structure of the flexfield.

The following examples demonstrate the code needed to accomplish these tasks. Example 22–17 and Example 22–18 apply to an ADF Desktop Integration implementation with the descriptive flexfield exposed as a static column. Example 22–19 presents the `isSegmentDisplayable()` method that is used in the other two examples.

Example 22–17 Updating or Inserting a Row with a Descriptive Flexfield Static Column

```
//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 descriptive flexfield row based on descriptive flexfield view accessor
  DFFViewRowImpl dffRow = (DFFViewRowImpl)srcRow.getAttribute(mydff_viewaccessor);

 //Check if single cell value is null
 if (dffAttributeValue != null && !("").equals(dffAttributeValue)) {
      //Getting DFF metadata information
      FlexfieldViewDefImpl dffImpl =
          (FlexfieldViewDefImpl)dffRow.getFlexfieldViewDef();
      String delim = dffImpl.getDelimiter();
      //Tokenizing DFF string
      StringTokenizer token =
          new StringTokenizer(dffAttributeValue.toString(), delim);

      while (token.hasMoreTokens()) {
          prjValues.add(token.nextToken());
      }
  }
  //Get descriptive flexfield segment information
  ListAttributeDef listSeg =
      dffRow.getFlexfieldViewDef().getFlexfieldAttributes();

  Iterator listSegIterator = listSeg.iterator();
  AttributeDef seg = null;
  ListString segDisplay = new ArrayListString();
  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 context dependent DFF case
 //If context is changed right before upload, 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 22–18 Applying Modified Segment Values to a Cell in a Descriptive Flexfield Static Column

You add this code as an application module method which will be invoked from 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 delimiter information for your DFF
 FlexfieldViewDefImpl dffImpl =
     (FlexfieldViewDefImpl)dffRow.getFlexfieldViewDef();
 String delim = dffImpl.getDelimiter();

//Get 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;
 Liststring segDisplay = new ArrayListstring();
 Listinteger segDisplayType = new ArrayListinteger();

 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 inbetween each segment value

  //  If the segment type is date, trim the time portion
  //  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) {
                        if (dffRow.getAttribute(segDisplay.get(i)) != null) {
                           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(dffRow.getAttribute(segDisplay.get(i)));
                  }
                                        }
             }else {
                 if (i==0)
                     segmentString.append(" ");
                  else {
                     segmentString.append(delim);
                     segmentString.append(" ");
                  }
             }
         }
   row.set your_transient_attribute(segmentString)
```

Example 22–19 isSegmentDisplayable() Helper Method Used in the Previous Examples

The input parameters for this method are the segment attribute definition and the descriptive flexfield row.

```
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;
            }
        }
    }
```

23

Using Key Flexfields

This chapter discusses how to use key flexfields in Oracle Fusion applications to access data that is represented by different organizations using different combinations of fields, and to customize the presentation of that information to end users in a way that is most appropriate for their organizations. This chapter also 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:

23.1 Introduction to Key Flexfields

A key flexfield is an intelligent key comprised of segments, in which one or more segments may have a meaning. An intelligent key, or *code*, uniquely identifies an object such as an account, an asset, a part, or a job, that implementers 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 you provide 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.

23.1.1 The 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 businesses use codes made up of meaningful segments (intelligent keys) to identify various business objects. For example, a company might have a part number "PAD-NR-YEL-8 1/2x14" indicating a notepad, narrow-ruled, yellow, and 14" by 8 1/2". A key flexfield lets you provide your users with a flexible *code* data structure that

users can set up however they like using key flexfield segments. Key flexfields let your users customize your application to show their codes any way they want them. For example, a different company might have a different code for the same notepad, such as "8x14-PD-Y-NR", and they can easily customize your application to meet that different need. Key flexfields let you satisfy different customers without having to reprogram your application.

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 only be available in certain colors.

23.1.2 How Key Flexfields are Modeled in Oracle Application Development Framework

Flexfields are modeled as a collection of Oracle Application Development Framework (ADF) polymorphic view rows. 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 should be used. Given a collection of polymorphic view rows, each row can be a different type.

The attribute sets associated with the discriminator are predefined. In fact, the Oracle Application Development Framework enables each view row to have its own view definition. When a polymorphic collection is created, the framework selects a view definition for the row to be added based on the value of the discriminator attribute.

With flexfields, this behavior is exposed using the terminology of segments and structures in place of attributes and view row types, respectively. The attributes in each view row definition are exposed as a set of segments with a predefined structure (or segment inclusion and ordering). The structure can include the discriminator attribute; common attributes that are unaffected by the discriminator value; and variable attributes that are included based on the discriminator value.

For key flexfields, the *structure instance number* (SIN) segment is the discriminator attribute, and the *code combination ID* (CCID) segment is the common attribute.

Flexfields use a context switching mechanism similar to that of polymorphic view objects. You use a wizard to generate a flexfield polymorphic view object based on the key flexfield definition, then create a view link to connect your product view object and the flexfield view object. You can then use the view object to add the flexfield to an application page.

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 Desktop Integration Developer's Guide for Oracle Application Development Framework*.

23.1.3 Partial Usage Feature

A key flexfield configuration can be shared with other application tables through the the *partial usage* feature. To share a configuration with another application table, you reuse the key flexfield definition over the application table by including one or all of the *master usage* segment columns in the application table. The table that contains the

redefined segment columns is referred to as a *partial table*. The master usage is sometimes referred to as *reference mode* and partial usage is referred to as *partial mode*.

There are two types of partial usages:

- *All-segment partial usage*: In this mode, the application table has columns for all of the key flexfield segments.
- *Single-segment partial usage*: In this mode, the application table has only one key flexfield segment column.

23.1.4 Participant Roles

As mentioned Section 21.2, "Participant Roles," this document uses the *owner* and *implementer* roles to clarify and group flexfield development activities. This chapter breaks the owner role into two categories:

Producer

The key 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. The producer's product owns the combinations table for that flexfield.

Consumer

A key flexfield *consumer* 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.

23.1.5 Completing the Key Flexfield Development Process

Before you start to incorporate key flexfields into your application, you need to determine whether you should complete the producer portion or the consumer portion of the key flexfield development process.

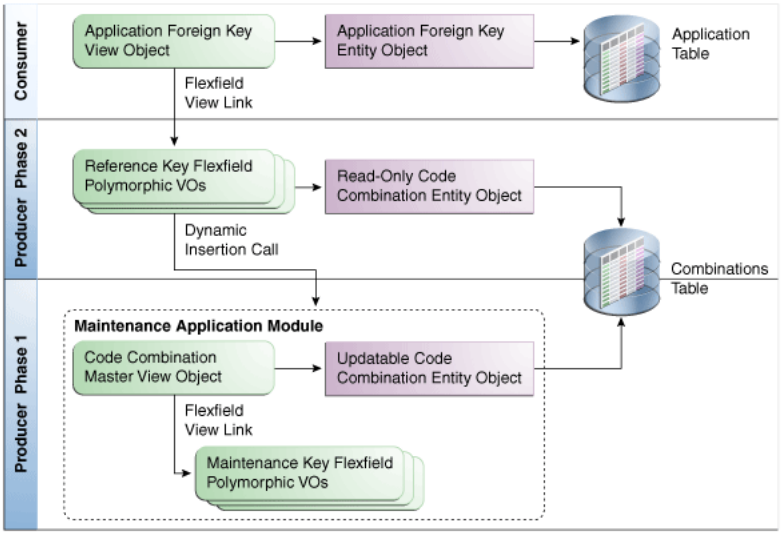
- If you have determined that a particular key flexfield is needed or would be useful within a particular application, and there is not yet a combinations table to support it, refer to the producer development tasks covered in Section 23.1.5.3, "Understanding the Key Flexfield Producer Development Tasks".
- If there is already a combinations table for the key flexfield that you want to implement, refer to the consumer development tasks covered in Section 23.1.5.4, "Understanding the Key Flexfield Consumer Development Tasks".

To incorporate key flexfield partial usages into your application, see Section 23.5, "Completing the Development Tasks for Key Flexfields in Partial Mode".

To employ key flexfield code combination filters in your application, see Section 23.6, "Working with Key Flexfield Combination Filters".

Figure 23–1 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. Section 23.1.5.3, "Understanding the Key Flexfield Producer Development Tasks" and Section 23.1.5.4, "Understanding the Key Flexfield Consumer Development Tasks" define the producer phases and summarize the steps for creating the components and artifacts shown in this figure. Note that the maintenance application module should be registered in the flexfield metadata if the key flexfield is configured for dynamic combination insertion.

Figure 23–1 Key Flexfield Development Roles, Business Components and Supporting Artifacts



Oracle Fusion Middleware Extensions for Applications provides key flexfield information such as combinations table structure. You can use this information to integrate your custom applications with key flexfields that are delivered with Oracle Fusion applications. For example, you can build foreign key pages that call an Oracle Fusion application's key flexfields.

23.1.5.1 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 code combination maintenance page enables application implementers 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 end users are not authorized to perform maintenance tasks directly. This is known as *dynamic combination insertion*.

  For example, when entering a transaction, a 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 behind the scenes.

The key flexfield producer builds the appropriate models to support maintenance mode and dynamic combination insertion.

23.1.5.2 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, as follows:

Cross Validation Rules

Cross validation rules leverage the code combination filter infrastructure to apply a pair of filters to new code combinations that are proposed for a key flexfield by administrators or end users.

At registration time, you need to 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 insert in C and PL/SQL.

23.1.5.3 Understanding 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 application table to support it, you need to 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 Section 23.2.1, "How to Develop Key Flexfields".

   Optionally, you can also do the following at registration time:

   - Implement segment labels.

     For more information, see Section 23.2.2, "How to Implement Key Flexfield Segment Labels".

   - Enable cross validation or register custom validation callouts.

     For more information, see Section 23.2.3, "How to Implement Cross Validation Rules and Custom Validation".

2. Create and configure the key flexfield business components.

   As shown in Figure 23–1, 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 over your combinations table and a master view object based on the updatable entity object. Next you create maintenance model key flexfield business components over the updatable entity object, and define a view link between your code combination 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 over your combinations table, and create reference model key flexfield business components over the read-only entity object.

See Section 23.2.4, "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 Section 25.4, "Regenerating Flexfield Business Components Programmatically".

3. Optionally, share your key flexfield business components with other developers using an ADF library.

   For more information, see Section 23.2.5, "How to Share Key Flexfield Business Components".

4. Optionally, build a key flexfield maintenance user interface.

   For more information, see Section 23.2.6, "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 a worksheet using ADF Desktop Integration.

   For more information, see Section 23.4, "Using Key Flexfield Advanced Features in Reference Mode".

6. Optionally, define, implement, and invoke key flexfield code combination filters.

   For more information, see Section 23.6, "Working with Key Flexfield Combination Filters".

23.1.5.4 Understanding the Key Flexfield Consumer Development Tasks

You can incorporate a producer's flexfields in your own application. For example, you might have an expenses table that references an account key flexfield in the General Ledger application.

If there are already columns in your application table to support the key flexfield that you want to implement, 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 a foreign key entity object and view object for your application.

To complete the consumer development tasks:

1. As shown in Figure 23–1, create a view link between your application's foreign key view object and the reference model key flexfield polymorphic view objects.

   See Section 23.3.1, "How to Create Key Flexfield View Links."

2. Nest the key flexfield application module instance in the application module for your application.

   See Section 23.3.2, "How to Nest the Key Flexfield Application Module Instance in the Application Module."

3. Add a key flexfield view object instance to the application module for your application.

See Section 23.3.3, "How to Add a Key Flexfield View Object Instance to the Application Module."

4. Add your key flexfield to an application page.

   See Section 23.3.4, "How to Employ Key Flexfield UI Components on a Page."

5. Configure the key flexfield user interface components.

   See Section 23.3.5, "How to Configure Key Flexfield UI Components."

After completing these tasks, you can define seed or test value sets for the flexfield, and you can create a model that you can use to test it. For more information, see Section 25.1.2, "How to Test Flexfields."

Once you have completed the key flexfield development process and delivered your application, implementers 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 implementers and administrators, you register it with *Oracle Fusion Functional Setup Manager*. For more information, see Section 25.5, "Integrating Flexfield Task Flows into Oracle Fusion Functional Setup Manager."

23.2 Completing the Producer Tasks for Key Flexfields

To prepare key flexfields for modeling in 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 partial usages, code combination filters, or the enabling of cross validation rules and custom validation callouts. All of these features require you to modify your application database.

Note: To incorporate a key flexfield partial usage into your application, you must have already defined and registered the key flexfield master usage on which it is based. See Section 23.2.1.5, "Registering and Defining Key Flexfields Using the Setup APIs", then continue to Section 23.5, "Completing the Development Tasks for Key Flexfields in Partial Mode".

To employ key flexfield code combination filters in your application, see Section 23.6, "Working with Key Flexfield Combination Filters".

The application table and its key flexfield columns must be registered in the Applications Core Data Dictionary before a flexfield can be defined on it.You accomplish this by deploying XDF files in the database. For more information, see Chapter 56, "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 Seed Data Loader.

After you complete the registration process described in Section 23.2.1, "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 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 implementer. In addition to the registration data, your flexfield seed data might include structures and value sets that you have defined for your flexfield.

For information about extracting and loading seed data, see Chapter 55, "Initializing Oracle Fusion Application Data Using the Seed Data Loader".

23.2.1 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.
3. Optionally, include the key flexfield segments in product tables for partial 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 partially reuse the key flexfield.
9. Register the entity details for each usage.

23.2.1.1 Creating the Combinations Table

Each key flexfield must have one corresponding table known as the *combinations table*. This table is also referred to as the *master table*.

> **Note:** The application table and its key flexfield columns must be registered in the Applications Core Data Dictionary before a flexfield can be defined on it. You accomplish this by deploying XDF files in the database. For more information, see Chapter 56, "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 SINs 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 U.S. and another for France). Each structure instance is identified by an 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. This is a way of adding simple striping to the combinations table. You insert the DSN column to enable you to tag sets of combination codes with your own numeric IDs, and 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 documentation.

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 Chapter 8, "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 23–1.

Table 23–1 Primary Key Configuration

Column	Include in the Primary Key
CCID	Always
SIN	When the flexfield is multiple structure–enabled or is multiple structure instance–enabled
DSN	When the flexfield is DSN-enabled

The combinations table must include the columns listed in Table 23–2. These columns indicate whether a combination is enabled and active. The column names and data types must match exactly.

Table 23–2 Required Combinations Table Columns

Column	Data Type	Description
ENABLED_FLAG	VARCHAR2(1) NOT NULL	A 'Y' value indicates that the combination is enabled. Any other value indicates that the combination is not enabled.
START_DATE_ACTIVE	DATE	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.
END_DATE_ACTIVE	DATE	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.

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 a 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 partial 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 Section 23.2.2, "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 be nullable or they should have default database values.

23.2.1.2 Creating Foreign Key Columns to Enable the Use of Flexfield Combinations on Application Pages

To permit the use of flexfield combinations on different application pages, you must include foreign key references to your combinations table's primary key configuration, as shown in Table 23–1, in other application tables. That way, you can display or enter valid combinations using forms not based on your combinations table. When you build a custom application that uses key flexfields, you include foreign key references in your custom application tables wherever you reference the flexfield.

> **Note:** Pages whose underlying entity objects contain a foreign key reference to the combinations table are referred to as *foreign key pages*, while pages whose underlying entity objects use the combinations table itself are referred to as *combinations pages* or *maintenance pages*.

23.2.1.3 Including Segment Columns in Partial 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 *partial* table. If a SIN or DSN is used, the partial table must either include those columns or a column from which the SIN or DSN can be derived.

23.2.1.4 Creating Filter Columns

You can use the key flexfield 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 Section 23.6, "Working with Key Flexfield Combination Filters."

23.2.1.5 Registering and Defining 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 about flexfield definitions.

The definition of a key flexfield includes the following information:

- The code, name, and description of the flexfield.
- 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 Section 23.2.1.1, "Creating the Combinations Table."

To learn how to access documentation about using the procedures in the following steps, see Section 23.2.1.6, "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 master 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 Section 23.2.1.9, "Registering Entity Details Using the Setup APIs." This step must be completed before you can generate the flexfield usage's business components.

23.2.1.6 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 outputs 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/applcore/db/sql/flex/fnd_flex_pkg_doc.sql \
FND_FLEX_KF_SETUP_APIS
```

23.2.1.7 Enabling Multiple Structure, Multiple Structure Instance, and Data Set Features

In order 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 access documentation about using the `enable_feature(...)` procedure, see Section 23.2.1.6, "What You May Need to Know About the Key Flexfield Setup API."

23.2.1.8 Partially Reusing a Key Flexfield on Another Table

Key flexfield partial usage enables you to capture the values of a key flexfield's segments in an application table. You can capture all of the flexfield's segments, or just one.

For information about partial reuse of a key flexfield, see Section 23.5, "Completing the Development Tasks for Key Flexfields in Partial Mode."

23.2.1.9 Registering 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 master usage, this is the entity object that was defined over the combinations table. For a partial usage, this is the entity object that was defined over the partial table.
- A prefix from which to derive the names of generated objects.
- The package in which to place the generated business components. Each usage can 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 Section 23.2.1.5, "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 access documentation about using the `create_adfbc_usage(...)` procedure, see Section 23.2.1.6, "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.

23.2.2 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 implementers and administrators should include when customizing the key flexfield. By defining segment labels when you define your key flexfield, you ensure that implementers customize the flexfield to include the segments that your application needs.

For example, the General Ledger application needs to 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, 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 implementer or administrator will define these key segments by defining two segment labels, GL_BALANCING and GL_ACCOUNT. When customizing your accounting flexfield, the implementer 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.

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 Section 23.4.2, "How to Access Segment Labels Using the Java API".

23.2.2.1 Defining Key Flexfield Segment Labels

You should define and register segment labels if you want to ensure that the application implementer or administrator customizes your key flexfield to include the segments that your application needs. For example, General Ledger defines "account" and "balancing" segment labels in the Accounting flexfield to ensure that implementers 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 implementer to tie it to at most one segment of the flexfield. You specify a segment label as `required` if you want the implementer 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 23–3 presents the results of setting these flags on a segment label in various combinations.

Table 23–3 Segment Label Flag Combinations

Global Flag	Required Flag	Unique Flag	Result
N	N	N	0+ (Zero or more segments)
N	N	Y	0,1 (Zero or one segment)
N	Y	N	1+ (One or more segments)
N	Y	Y	1 (Exactly one segment)
Y	-	-	ALL (All segments; global flag overrides the other flags)

For example, in General Ledger's Accounting flexfield, the **Account** segment label is required and unique because 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 Section 23.2.1.5, "Registering and Defining Key Flexfields Using the Setup APIs."

To learn how to access documentation about using the `create_segment_label(...)` procedure, see Section 23.2.1.6, "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.

23.2.2.2 Using 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.

You 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 Section 23.2.1.5, "Registering and Defining Key Flexfields Using the Setup APIs."
2. Define the segment label as described in Section 23.2.2.1, "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.

23.2.3 How to Implement Cross Validation Rules and Custom Validation

You use procedures from the FND_FLEX_KF_SETUP_APIS PL/SQL API 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 for the flexfield, so new code combinations entered on a 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
3. Customer custom validation callout

23.2.3.1 Implementing Cross Validation Rules

To implement a cross validation rule for a key flexfield, you 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 you build a maintenance user interface that administrators can use to maintain their own rule definitions.

Before you begin:

Before you can build a cross validation rule maintenance user interface, you must first have created and configured the business components for the key flexfield to which the rule will apply.

For more information, see Section 23.2.4, "How to Create Key Flexfield Business Components."

To learn how to access documentation about using the FND_FLEX_KF_SETUP_APIS PL/SQL package, see Section 23.2.1.6, "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 whether 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 23–4.

Table 23–4 FND_KF_CROSS_VAL_RULES Cross Validation Repository Table

Column	Type	Nullable?	Description
ENTERPRISE_ID	NUMBER(18)	No	(PK) Enterprise ID.
STRUCTURE_INSTANCE_ID	NUMBER(18)	No	(PK) Structure Instance ID.
RULE_CODE	VARCHAR2(30)	No	(PK) Developer key for this rule.
DESCRIPTION	VARCHAR2(240)	Yes	Rule description.
CONDITION_FILTER	XMLTYPE	Yes	Flexfield filter defining where the rule should be applied. NULL means globally applied.
VALIDATION_FILTER	XMLTYPE	Yes[1]	Flexfield filter defining the validation that must be true.
ERROR_MSG_APPLICATION_ID	NUMBER(18)	Yes	Message application.
ERROR_MSG_NAME	VARCHAR2(30)	Yes	Message to display if rule is violated. If NULL, display default message.
ENABLED_FLAG	VARCHAR2(1)	No	Y/N
START_DATE_ACTIVE	DATE	Yes	Standard start date.
END_DATE_ACTIVE	DATE	Yes	Standard end date.
WHO Columns	WHO	WHO	Standard WHO Columns.

[1] Although the validation filter must be made nullable in the data model, it is still a required value that is logically necessary for cross validation to work.

The primary key for this table is the combination of ENTERPRISE_ID, STRUCTURE_INSTANCE_ID, and RULE_CODE.

The cross validation rule itself is the combination of a condition filter and a validation filter in the corresponding XMLType columns of the repository table. These filters are compatible with, and supported by, the key flexfield combination filter infrastructure.

The value of each of these filters should be a logical combination of boolean expressions. At runtime, all filters from this 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 the pseudocode in Example 23–1.

Example 23–1 Applying a Cross Validation Rule

The condition filter specifies a value range for one segment:

```
segment1 <= '10'
```

If the condition is met, the validation filter is applied:

```
(segment2 = '20') OR (segment2 = '30')
```

If the proposed code combination is three segments with the following values, the validation will succeed:

```
segment1 = '8'
segment2 = '30'
segment3 = '70'
```

If the proposed values are as follows, the condition is not met, and the code combination will not be subject to the validation filter:

```
segment1 = '12'
segment2 = '40'
segment3 = '70'
```

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 NULL, all new code combinations for the flexfield will qualify to be evaluated with the validation filter.

You use the key flexfield combination filter infrastructure to create a separate maintenance page for each key flexfield that supports cross validation rules.

For more information, see Section 23.6.2, "How to Add Combination Filters to Your Application", Section 23.6.6, "How to Remove Combination Filters from Your Application", and Section 23.6.3, "How to Employ Combination Filters on an Application Page".

23.2.3.2 Implementing Custom Validation

To implement custom validation, you register a PL/SQL validation procedure.

To implement custom validation with the custom validation callout:

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 23–2.

Example 23–2 PL/SQL Validation Procedure Signature

```
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 in the combinations table that are not related to the flexfield will be passed as null.

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, 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 dynamic insert will be aborted, and the message in the exception displayed to the user. The log will also record the entire call stack, including the fact that the exception was raised from a custom callout. If the API returns without exception it will be considered a success.

Once you have written the custom validation callout procedure, you can register it with the key flexfield. The custom validation callouts are registered in the FND_KEY_FLEXFIELDS_B key flexfield registration table as shown in Table 23–5.

Table 23–5 Key Flexfield Custom Validation Callouts

Column	Type	Nullable?	Description
DEVELOPER_VAL_CALLOUT	VARCHAR2(80)	Yes	PL/SQL validation callout procedure for development use.
CUSTOMER_VAL_CALLOUT	VARCHAR2(80)	Yes	PL/SQL validation callout procedure for customer use.

To learn how to access documentation about using the FND_FLEX_KF_SETUP_APIS PL/SQL package, see Section 23.2.1.6, "What You May Need to Know About the Key Flexfield Setup API."

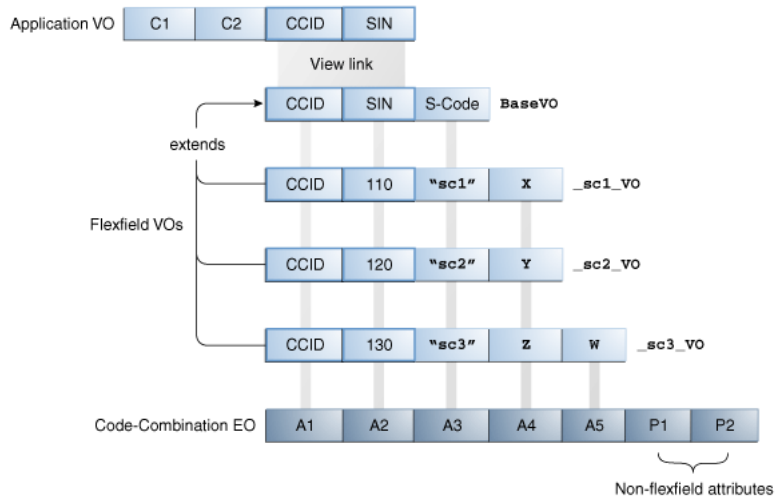
23.2.4 How to Create Key Flexfield Business Components

You need to define view objects based on each key flexfield combinations table. The base key flexfield 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.

Figure 23–2 shows a sample configuration of an application view object using 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 application view object.

Figure 23–2 Key Flexfields Modeled as ADF Business Components



If the combinations table has other fixed (non-flexfield) columns, they are not included in these view objects.

No Java implementation classes are generated for key flexfield view objects. The application 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 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.

   You can build this model in one of the following ways:

   - Without dynamic combination insertion support.

     To accomplish this, you simply 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 application 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 standard wizard, create application entity objects based on the combinations tables you have defined. Make sure of the following:

- At least one customization class is included in `adf-config.xml`. 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 *Oracle Fusion Applications Extensibility Guide*.
- These entity objects are directly modeled on the combinations tables; hence they contain the fixed (non-flexfield) 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 Section 23.2.1.9, "Registering Entity Details Using the Setup APIs".

23.2.4.1 Building 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, manage combination locking to override the default automatic combination locking with your own implementation.

23.2.4.1.1 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 be sure to 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 over your combinations table and add it as an ADF Business Components usage for your key flexfield, as described in Section 23.2.1.9, "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 `oracle.apps.fnd.applcore.oaext.model.KFFMEntityImpl`, and the entity definition class must extend `oracle.apps.fnd.applcore.oaext.model.KFFMEntityDefImpl`.

> **Caution:** You should disable the delete capability for the code combinations table, as the deletion of previously created combinations might invalidate foreign key references. If you want to disallow the use of a combination, you should disable the combination instead of deleting it.

2. Create a master view object 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`.

To create key flexfield business components:

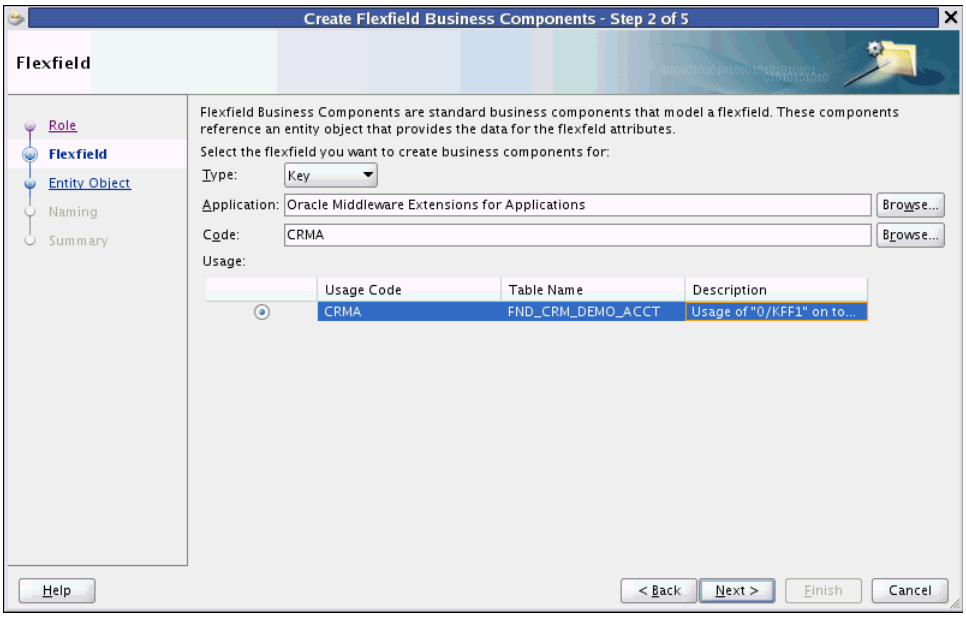
1. Build your project to ensure that the entity objects are available in classes. The modeler relies on what is in your classes.
2. In the New Gallery, navigate to **Business Tier** > **ADF Business Components** and select **Flexfield Business Components**.
3. Click **OK** to access the Create Flexfield Business Components wizard.
4. On the Role page, select the role that you are taking as you create the flexfield business components:
   - **Developer** — select this role if you are incorporating the flexfield into an application. The business components must be stored in one of your projects. Select the desired project location from the **Project Source Path** dropdown list.
   - **Tester** — select this role if you are planning to test or share your flexfield. In the **Output Directory** field, specify the path of your desired location for the generated business components.

     For more information about testing flexfields, see Chapter 25, "Testing and Deploying Flexfields". For more information about sharing and importing shared flexfields, see Section 23.2.5, "How to Share Key Flexfield Business Components".

   > **Note:** This is not a role in the security sense. It exists only during this procedure, for the purpose of specifying where your generated flexfield business components should be stored.

5. Click **Next**. The Flexfield page appears, as shown in Figure 23–3.

Figure 23–3 Create Flexfield Business Components Wizard — Flexfield Page



6. From the **Type** dropdown list, select **Key**.
7. In the **Application** field, specify the full name of the application to which your key flexfield belongs.

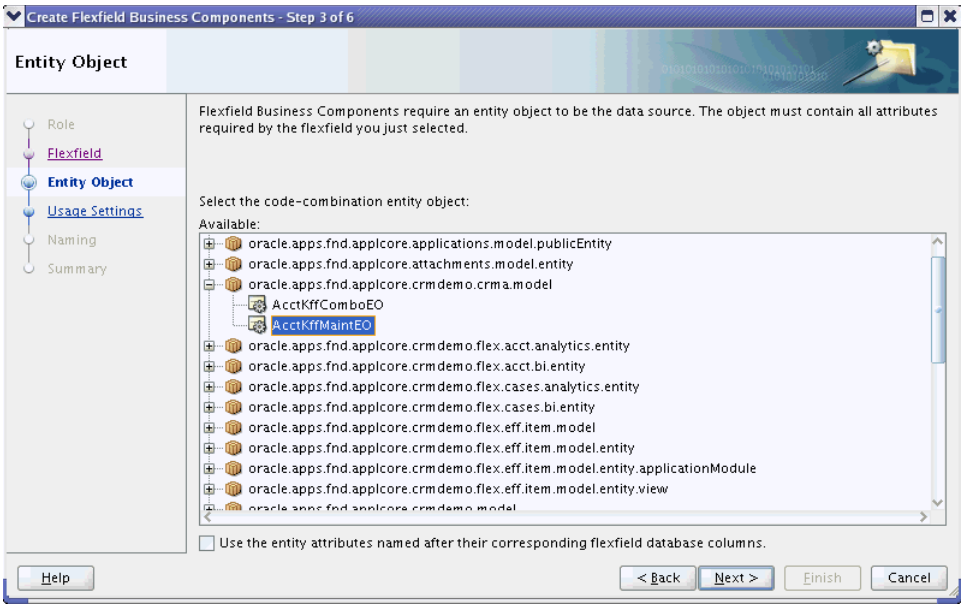
   You can browse for the name, and filter by **ID**, **Short Name**, or **Name**.
8. In the **Code** field, specify the code of the key flexfield you want to use.

   You can browse for and filter by **Code**.
9. In the **Usage** section, select the table row that contains the master (primary) usage of the key flexfield as it is defined on its combinations table. Every key flexfield has exactly one master usage.

   To identify the master 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 **Description** field does *not* contain the prefix `(Partial)` or `(Partial Single)` in parentheses.

   You must select the master usage in this procedure because you are generating key flexfield business components over your combinations table for your maintenance model.
10. Click **Next**. The Entity Object page appears, as shown in Figure 23–4.

Figure 23–4 Create Flexfield Business Components Wizard — Entity Object Page



11. Expand the tree of available models and select an entity object to use as the data source for the key flexfield.

    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 master key flexfield usage, this includes attributes that represent the CCID, SIN, and segment columns, and the DSN column if it exists in the combinations table.

12. 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, select the checkbox labeled **Use the entity attributes named after their corresponding flexfield database columns**. This checkbox is unselected by default.

    When a key flexfield entity object attribute is transient, there is no matching underlying column name. When you select this checkbox, 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. Make sure 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 base table usage. There is no need to register another table for this purpose, even if the entity object is based on some other table. See Section 23.2.1.9, "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 upper case.

13. Click **Next**. The Usage Settings page appears.

    Because you specified the master 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.

14. Click **Next**. The Naming page appears.

    To create business components, the package name and the object name prefix for the selected entity object must first be registered with the key flexfield master usage. Text on the Naming page indicates whether this is the case:

    - If the selected entity object is registered with the flexfield usage, the Naming page displays the package name and the object name prefix for the entity object. Click **Next** and continue to Step 15.
    - If the selected entity object is not registered as an ADF Business Components usage, the Naming page displays a message to that effect. Take one of the following actions:
      - Click **Back** to return to the Entity Object page and select an entity object that has been properly registered.
      - Click **Cancel** to exit this wizard and register the entity object that you want to use as described in Section 23.2.1.9, "Registering Entity Details Using the Setup APIs."

15. 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 application 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.

16. Refresh the project to see the newly created flexfield business components in the Application Navigator.

23.2.4.1.2 How to Link Your Maintenance Model Key Flexfield Business Components to Your Code Combination Master View Object

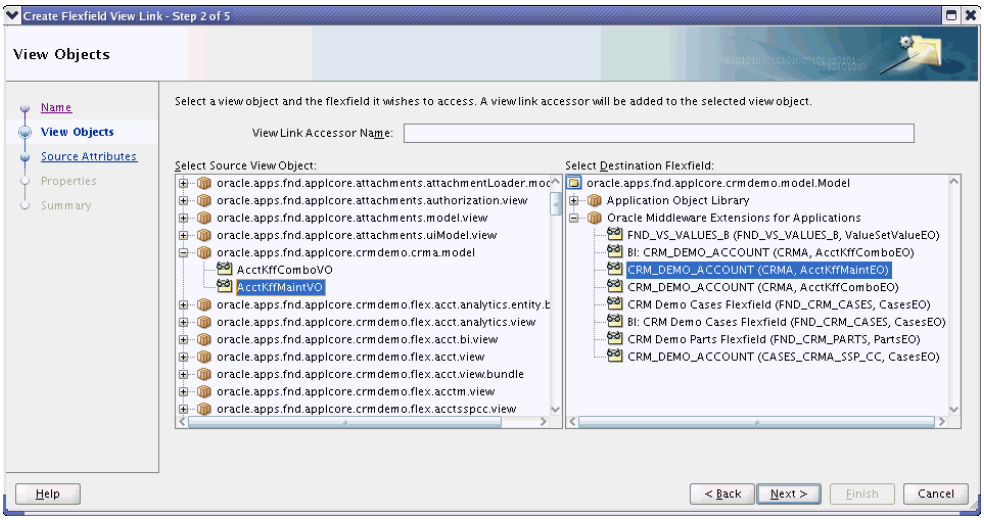
You need to create a flexfield view link from your code combination master view object to the maintenance key flexfield business components. This enables your maintenance user interface to access all of the combinations table columns using the linked view objects over the combinations table entity object.

The master view object and the base key flexfield view object are linked through the combination of a CCID, and SIN, and if present, a DSN.

To create a key flexfield maintenance model view link:

1. In the New Gallery, navigate to **Business Tier** > **ADF Business Components** and select **Flexfield View Link**.
2. Click **OK** to access the Flexfield View Link wizard.
3. On the Name page, from the **Package** dropdown list, specify a package for the view link.
4. In the **Name** field, enter a name for the view link.
5. Click **Next**. The View Objects page appears, as shown in Figure 23–5.

Figure 23–5 Create Flexfield View Link Wizard — View Objects Page



6. In the **Select Source View Object** tree, expand the available objects from your current project and select your code combination master view object.
7. 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.
8. In the **View Link Accessor Name** field, enter an appropriate name for the view link accessor.
9. 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 Section 23.2.4.1, "Building a Writable Maintenance Model" and recreate your maintenance mode business components according to the instructions.

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**.

23.2.4.1.3 How to Create the Maintenance Application Module

You need to 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 Section 23.2.4.1.1, "How to Create Key Flexfield Business Components for a Maintenance Model."

For more information about creating application modules and nesting application model instances, see the "Implementing Business Services with Application Modules" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

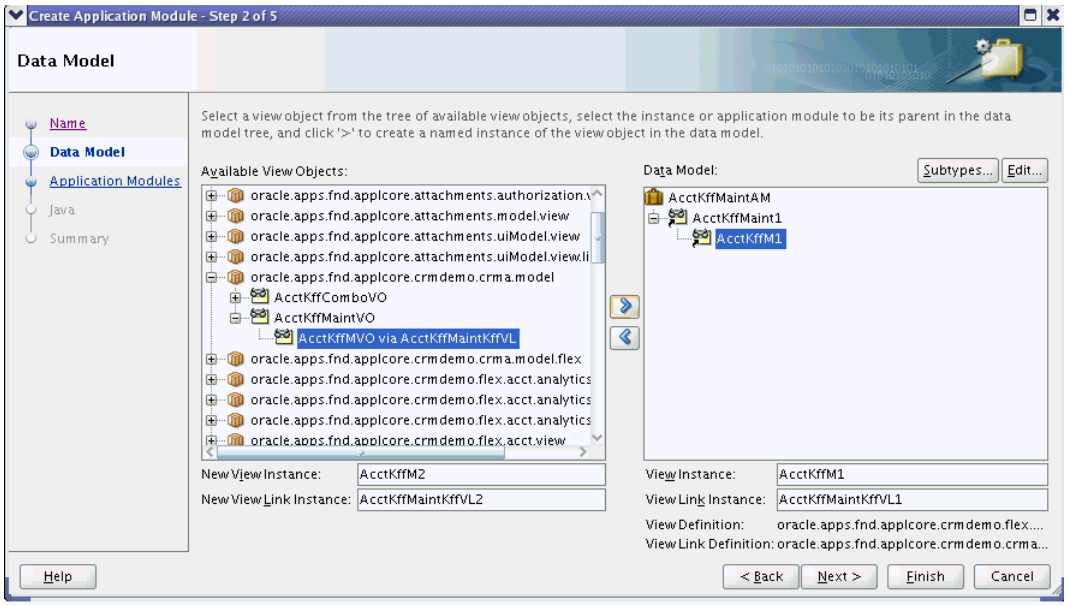
Before you begin:

1. Create the business components as described in Section 23.2.4.1.1, "How to Create Key Flexfield Business Components for a Maintenance Model."
2. Create the view link as described in Section 23.2.4.1.2, "How to Link Your Maintenance Model Key Flexfield Business Components to Your Code Combination Master View Object."

To create the maintenance application module:

1. Use the standard wizard to create an application module.
2. In the Data Model Components page, shuttle the combination master view object and the maintenance model view link to the **Data Model** list, as shown in Figure 23–6.

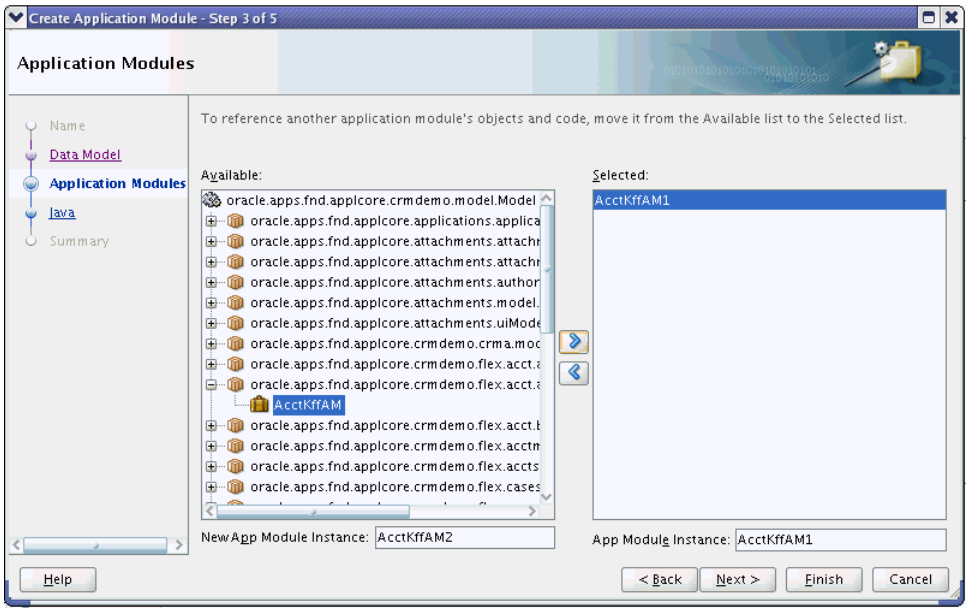
Figure 23–6 Create Application Module Wizard — Data Model Page



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.

3. Click **Next**. The Application Modules page appears.
4. On the Application Modules page, in the **Available** tree, shuttle the appropriate key flexfield application modules to the **Selected** list, as shown in Figure 23–7. Be sure to include the key flexfield application module that was created when you created the business components in Section 23.2.4.1.1, "How to Create Key Flexfield Business Components for a Maintenance Model."

Figure 23–7 Create Application Module Wizard — Application Modules Page



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 project application module.

5. When you complete the Create Application Module wizard, right-click the new application module instance and choose **Run** to test it.

Note: The master application module must have a configuration named `<appmodule_name>Shared`. By default, this is created for you. For example, if the application module is called `MyKffMaintAM`, then a configuration named `MyKffMaintAMShared` 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 Section 23.4.4, "How to Publish Key Flexfield Application Modules as Web Services."

23.2.4.1.4 How to Manage Combination Locking

Automatic combination locking is enabled by default in `oracle.apps.fnd.applcore.oaext.model.KFFMEntityImpl`, which is extended by your code combination entity object class. The method `doDML(int, TransactionEvent)` is overridden to lock the 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, refer to

the Java documentation of `oracle.apps.fnd.applcore.oaext.model.KFFMEntityImpl`.

23.2.4.2 Enabling Dynamic Combination Insertion

This task is necessary only if you want to permit end users to create new code combinations extemporaneously on an application page. You must have already built a writable key flexfield maintenance model. For more information, see Section 23.2.4, "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, craft 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.

23.2.4.2.1 Enabling 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 master 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 Section 23.2.1.6, "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, `oracle.apps.fnd.applcore.flex.test.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 Section 23.2.1.6, "What You May Need to Know About the Key Flexfield Setup API."

23.2.4.2.2 Inserting a Code Combination — the Simplest Case In the simplest case, you need only replace the existing base object class, `oracle.apps.fnd.applcore.oaext.model.OAApplicationModuleImpl`, with

`oracle.apps.fnd.applcore.oaext.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.
- You have only one nested key flexfield application module instance in this application module, and this 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.

23.2.4.2.3 Inserting a Code Combination with Added Combination Attributes To insert a code combination with added combinations, you implement the Java class `oracle.apps.fnd.applcore.oaext.model.KFFCombinationCreator` in the master application module, and you create a Java implementation of the maintenance application module. You can optionally initialize the columns.

1. In the master application module, implement the Java class `oracle.apps.fnd.applcore.oaext.model.KFFCombinationCreator`.

   The class has only one method defined:

   ```
   public 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 `oracle.apps.fnd.applcore.oaext.model.OAApplicationModuleImpl`. An example of an application module class is shown in Example 23–3.

Example 23–3 Implementing KFFCombinationCreator

```
public class MyKffMaintenanceAM
  extends OAApplicationModuleImpl
  implements KFFCombinationCreator
{
  /**
   * Container's getter for MyKffAM1.
   *
   * @return MyKffAM1
   */
  public ApplicationModuleImpl getMyKffAM1()
  {
    return (ApplicationModuleImpl) findApplicationModule("MyKffAM1");
  }

  public void createKeyFlexfieldCombination(
```

```
  Long sin, Long dsn, List<Object> segValues)
  {
    KFFCombinationAttributes combAttrs =
      ((KFFMApplicationModuleImpl)
      getMyKffAM1()).insertCombination(sin, dsn, segValues);
  }
}
```

By default, JDeveloper creates accessor methods (such as `getMyKffAM1()`) to all of your nested application modules. You can make use of these accessor methods to access the key flexfield application module, as shown in bold in Example 23–3.

If you do not need to update any combination attributes, 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 23–4 demonstrates how to use the `KFFCombinationAttributes` object to access the value-attribute values and update the value attribute columns of the combinations table.

Example 23–4 Using the KFFCombinationAttributes Object

```
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_VALUE_ATTRIBUTE1";

// Get the current combination value attribute.
System.out.println(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).
}
```

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 the following example, an alternative version of `MyKffMaintenanceAM.java`.

Example 23–5 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);
    }
}
```

23.2.4.2.4 Inserting 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 `oracle.apps.fnd.applcore.oaext.model.KFFCombinationCreatorProxy`. This makes the information in the `KFFCombinationCreatorProxy.Context` object, such as validation date, available to the custom validation procedures.

1. In the master application module, generate a Java class that implements `oracle.apps.fnd.applcore.oaext.model.KFFCombinationCreatorProxy` and extends your existing base object class. By default, the base class is `oracle.apps.fnd.applcore.oaext.model.OAApplicationModuleImpl`. An example of an application module class is shown in Example 23–3.

> **Note:** `KFFCombinationCreatorProxy` is a sub-interface of `KFFCombinationCreator`.

Example 23–6 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);
  }
}
```

2. Optionally, initialize value attributes, as described in Step 3 of Section 23.2.4.2.3, "Inserting a Code Combination with Added Combination Attributes."
3. Optionally, initialize other columns of the combinations table, as described in Step 4 of Section 23.2.4.2.3, "Inserting a Code Combination with Added Combination Attributes."

23.2.4.3 Building 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 page with a foreign key reference to the code combinations table.

Note: If you want to permit end users to create new code combinations extemporaneously 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 Section 23.2.4, "How to Create Key Flexfield Business Components."

Before you begin:

Create a read-only entity object over your combinations table and add it as an ADF Business Components usage for your key flexfield, as described in Section 23.2.1.9, "Registering Entity Details Using the Setup APIs."

To create key flexfield business components for a read-only reference model:

1. Complete Steps 1 through 10 in Section 23.2.4.1.1, "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 master key flexfield 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 Steps 12 through 16 in Section 23.2.4.1.1, "How to Create Key Flexfield Business Components for a Maintenance Model."

23.2.5 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

23.2.5.1 Creating an ADF Library JAR File

If you are the owner of the flexfield business components you want to share, 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 New gallery, select **General > Deployment Profiles > ADF Library JAR File** and click **OK**.

   b. 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.

23.2.5.2 Importing Business Components From an ADF Library

Once 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 *Oracle Fusion Middleware Fusion Developer's Guide for 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**.
3. Select **Business Components > Imports**.
4. Click **Import**, then navigate 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.

23.2.6 How to Build a Key Flexfield Maintenance User Interface

You can 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.

Before you begin:

Create a maintenance application module over the combinations table and a writable maintenance model for the flexfield usage as described in Section 23.2.4.1, "Building a Writable Maintenance Model."

To build a key flexfield maintenance page:

1. From the Data Controls panel, expand the maintenance application module that you created over 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 into the form or table that you just created.

For more information, see Section 23.3.4, "How to Employ Key Flexfield UI Components on a Page".

23.2.7 What Happens at Runtime: Creating New Combinations

At runtime, an instance of the registered application module is created. Whenever a new combination needs to 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.

23.3 Completing 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 master usage (also referred to as *reference mode*) are:

1. Create a view link from your (consumer) application 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.
4. Add your key flexfield to an application page.
5. Configure the flexfield user interface components.

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 Section 23.3.5.1, "Configuring Flexfield-Level User Interface Properties."

23.3.1 How to Create Key Flexfield View Links

A view link is needed whenever one of your application view objects references the producer's key flexfield. The application view object and the base key flexfield view object are linked through the combination of a CCID, an SIN, and if present, a DSN. The key flexfield view object can have many incoming view links from various application view objects, as a key flexfield is usually referenced by many application tables. For example an ExpenseLines view object might have a foreign key reference to the GLKff view object.

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 application table that references the combinations table. This means that when a user does a search on the application 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 application table.

Before you begin:

- Ensure that the flexfield library jars that were created by the producer team have been added to your project. For more information about library jars, see Section 23.2.5.1, "Creating an ADF Library JAR File" and Section 23.2.5.2, "Importing Business Components From an ADF Library."
- You should have already created a foreign key entity object and view object for your foreign key application table that references the key flexfield.

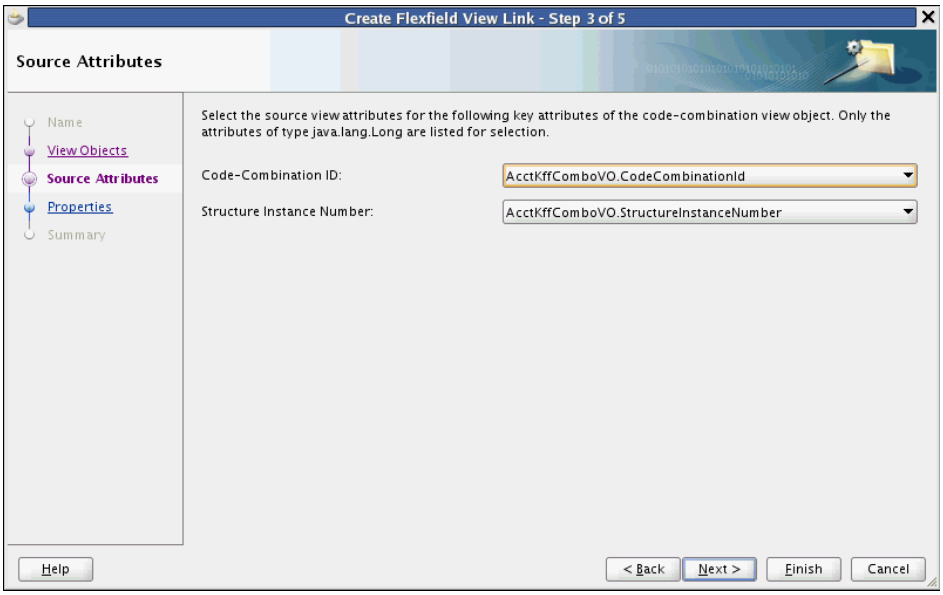
  Ensure that the view object does not include flexfield attributes such as **SEGMENT1_VARCHAR2**, **SEGMENT2_NUMBER**, and so on. Ensure that you include the attributes that are needed for the foreign key reference, such as **CCID**, **SIN**, and, if present, **DSN**. Ensure that the CCID attribute's **Display** control hint is set to `Hide`.

To create a key flexfield view link:

1. In the New Gallery, navigate to **Business Tier** > **ADF Business Components** and select **Flexfield View Link**.
2. Click **OK** to access the Flexfield View Link wizard.
3. On the Name page, from the **Package** dropdown list, specify a package for the view link.
4. In the **Name** field, enter a name for the view link.
5. Click **Next**. The View Objects page appears.
6. In the **Select Source View Object** tree, expand the available objects from your current project and select a source view object.

7. In the **Select Destination Flexfield** tree, expand the available flexfield view objects from your project and select a destination base key flexfield 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 23–8.

Figure 23–8 Create Flexfield View Link Wizard — Source Attributes Page for Key Flexfields



10. 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`.

11. 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.

12. 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`.

13. Click **Finish** to go to the Summary page.

Note: You can skip the Properties page because view link-specific properties are not supported.

14. On the Summary page, review the summary, then click **Finish** to create the view link.
15. Optionally, disable the automatic propagation of value set security rules to the application table by adding a custom property to the view link between the application view object and the key flexfield view object, as follows:

```
<propertyname="FND_ACFF_MasterSecuredByFlexfield" Value="false"/>
```

23.3.2 How to Nest the Key Flexfield Application Module Instance in the Application Module

You use the overview editor for your application module to nest the key flexfield application module instance. 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 an application module for your application.

To Nest the Key Flexfield Application Module Instance in the Application Module

1. In the Application Navigator, double-click the root 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, shuttle the key flexfield application module to the **Selected** list.

23.3.3 How to Add a Key Flexfield View Object Instance to the Application Module

You need to add a flexfield view object instance that reflects the hierarchy of the view link that you created in Section 23.3.1, "How to Create Key Flexfield View Links" to the application module for your application. 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 your foreign key application 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, ExpenseLines is the master and GLKff is the detail.

For more information about creating a hierarchy of view instances, see "Adding Master-Detail View Object Instances to an Application Module" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

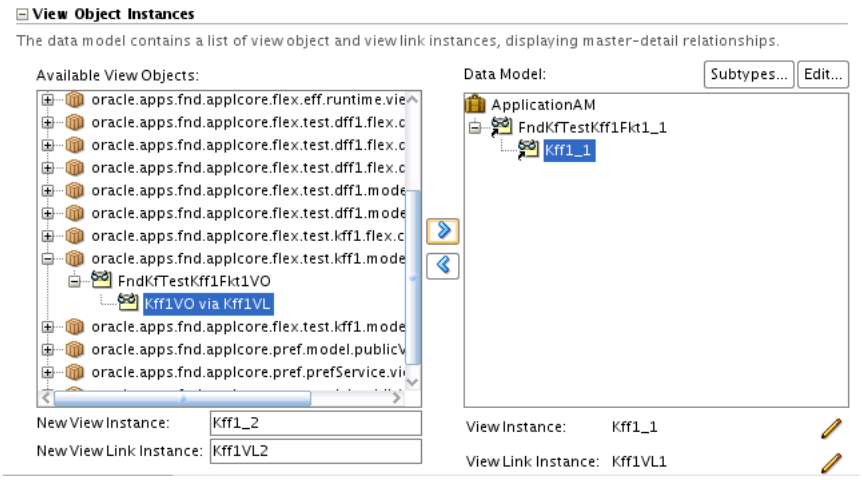
Before you begin:

1. Add an instance of the view object for your foreign key application table to the application module for your application, as described in the "How to Add a View Object to an Application Module" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
2. Create a view link between the view object for the foreign key application table and the key flexfield view object as described in Section 23.3.1, "How to Create Key Flexfield View Links."

To Add a Key Flexfield View Object Instance to the Application Module

1. In the Application Navigator, double-click the application module for your application.
2. In the overview editor, click the **Data Model** navigation tab.
3. Expand the **View Object Instances** section.
4. If an instance of the view object for your foreign key application table does not appear in the **Data Model** list, select it in the **Available View Objects** list and shuttle it over. This is the master view object instance.
5. In the **Data Model** list, select the instance of the view object for your foreign key application table (the master view object instance) so that it appears highlighted. This will be the target of the detail flexfield view instance you will add.
6. In the **Available View Objects** list, expand the view object for your foreign key application table and shuttle 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.

Figure 23–9 Flexfield View Instance Nested Under Master View Instance



23.3.4 How to Employ Key Flexfield UI Components on a Page

To employ a key flexfield UI component on an application page, you 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**

  The base 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 containing code combination IDs.

  The primary purpose of foreign key 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 foreign key pages that use a given key flexfield.

- **A page with partial usage of a key flexfield**

  You can invoke the *partial usage* feature of key flexfields on a page. Partial usage occurs when one or all segments of a key flexfield that have already been defined over a combinations table are redefined over a product table. In this way you can reuse a key flexfield definition over a transactional table as if it is a descriptive flexfield.

- **A code combination maintenance page**

  The only purpose of a code combination maintenance page (often referred to as a *combinations 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 Section 23.2.6, "How to Build a Key Flexfield Maintenance User Interface."

  A typical application has one and only one combinations page. An application might not have a combinations 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. End users can select which attributes of the key flexfield view object to use as criteria. See Section 23.3.6, "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 combinations 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 pages with foreign key references to the same key flexfield. For example, in an order entry/inventory application, you might have a

combinations page where you define new parts with a key flexfield for the part numbers.

You would also have a page with a foreign key reference 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 key flexfield combination filter, which you use to determine the acceptable values of your part numbers. This combination filter references the same key flexfield as the combinations page and the foreign key 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 partial mode SIN values
- For partial flexfield segments or segments on a combinations 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.

23.3.4.1 Adding 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 not 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.

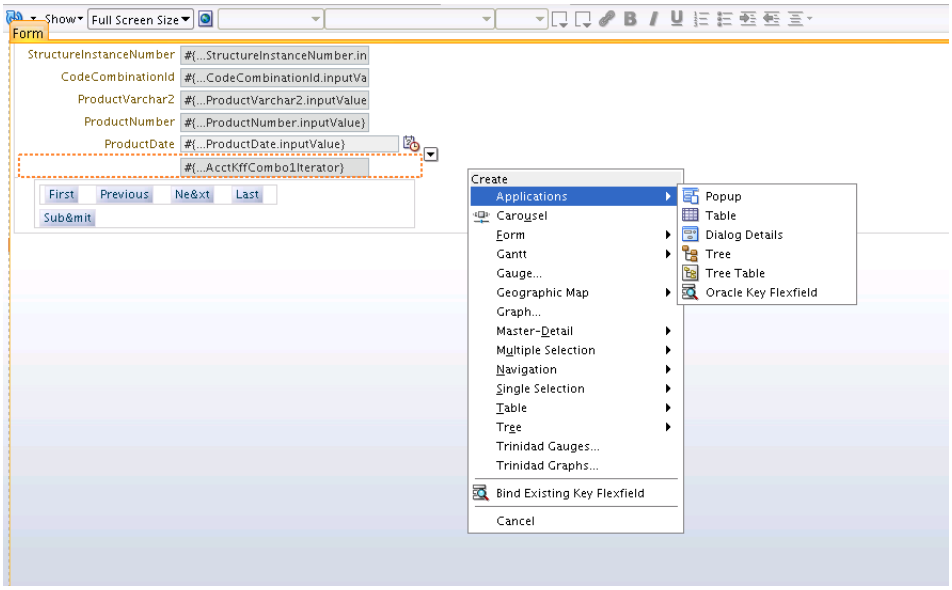
2. When prompted, select the type of user interface that you want to create:

   - ADF Form
   - Applications > Panel
   - ADF Table
   - Applications > Table

   For an ADF Table component, 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 into a form or a table, and select the appropriate flexfield UI component. Figure 23–10 shows a key flexfield being dropped into a form.

Figure 23–10 Key Flexfield Dropped into a Form



4. Using either sequence generation or default values, ensure that when the 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 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, select **editAll**. If you want the user to click into a row to make it editable, select **clickToEdit**.

   If you select **clickToEdit**, the editable row displays the concatenated flexfield segment values in an input text component. The 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.

23.3.4.2 Ensuring Proper Handling of New Rows

When a new row that contains key flexfield columns is added on an application 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 in reference mode, end users will not be able to select key combinations for the new row. Without default structures in partial mode, 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 foreign key entity object Structure Instance Number attribute or the foreign key view object Structure Instance Number attribute, 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:** For partial flexfields in table components, you can use the `defaultSIN` attribute in the JSPX file to define the default Structure Instance Number value for situations where no rows exist.

For an Applications Table component in reference mode, the default Structure Instance Number 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 partial usages, the Structure Instance Number value of the first row of a user interface table that contains partial mode columns determines the partial mode column structure to be used for the table. For any additional row that contains a different Structure Instance Number, the partial mode 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 partial mode column structure.
>
> For an Applications Table component, if the end user deletes all rows from the table, your application can once again set a new default Structure Instance Number value for the first new row, and the partial mode column structure corresponding to that Structure Instance Number will be the valid structure for the table.
>
> For an ADF Table component, the partial mode 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

23.3.4.3 Ensuring 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 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, the developer 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 user to set the CCID to null, change the SIN value, and then enter the CCID value in order for the concatenated value to be updated.

23.3.4.4 Ensuring Proper Updating of Partial Mode SIN Values in an ADF Form

When an end user makes changes to an existing ADF Form row that contains key flexfield partial 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 23–7.

Example 23–7 Code for Updating Modified SIN Values

```
//Add the partial target for the parent of the key flexfield partial usage
    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 partial 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 component or an Applications Table component. Dynamically changing the SIN at runtime is supported only for an ADF Form component.

23.3.4.5 Dynamically Refreshing Partial Flexfield Segments and Segments on a Combinations Page

If you have segments on a combinations page or partial flexfield segments, and those segments are in an ADF Table component that is wrapped in an Applications Table component that is refreshed by another component, such as a button or a search query, 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 23–8 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 23–9 shows the binding of the custom flexfield handler to the query component.

Example 23–8 Flexfield Listener

```
public void customKffSearchQueryListener(QueryEvent queryEvent)
{
  invokeMethodExpression(
    "#{bindings.KffCriteriaQuery.processQuery}",
    Object.class,QueryEvent.class,queryEvent);
  DescriptiveFlexfield.updateFlexColumns(appTable);
}
```

Example 23–9 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 refresh for standard Applications Table create and delete operations. However, custom create and delete operations must handle the refreshing of flexfields.

23.3.4.6 What Happens When You Add a Key Flexfield to a Page

Key flexfields are implemented in the user interface as code combination LOVs rather than as individual segments on the page. You can type a combination code directly into the code combination LOV input.

Figure 23–11 shows an example of a key flexfield used in a form on an application page:

Figure 23–11 Example of a Key Flexfield In a Form

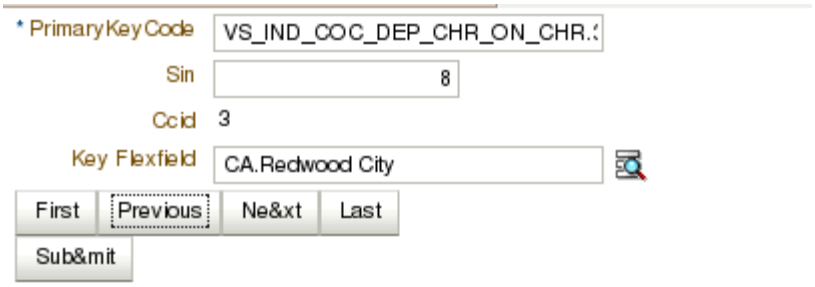


Figure 23–12 shows an example of a key flexfield used in a table on an application page:

Figure 23–12 Example of a Key Flexfield in a Table

	PartNumber	Description	UnitCost	Sin	Account	CRM Demo Parts Flexfield		
						Inspection Required	Inspection Date	Lead Time
▷	3721962	Plastic Wheels	2.25	12206 CRM_DEMO_	AB10.000	Y	1/1/09	1
▷	3722508	Padded Seat	39.99	12207 CRM_DEMO_ORG_A	DEV.AB10.000	N		6
▷	3722781	Executive Seat Back	59.99	12208 CC_ACCT_LOC_DIV_	DG31.000.6111.0000	Y	1/2/09	2
▷	3723054	Standard Seat Back	29.99	12209 ORG_CC_ACCT_LO	DEV.AB10.000.4000.0000	Y	1/3/09	3
▷	3723327	Chair Stand	15.25	12210 ORG_CC_ACCT_LO	SALES.DG31.001.4211.5711.00000	N		12
▷	3723600	SI Supercomputer Case	99.99	12211 ORG_CC_ACCT_LO	HR.AB10.000.3111.0000.00000.0000	N		15
▷	3723873	SI Supercomputer Mothert	399.99	12212 CC_ACCT_LOC_DIV_	CZ20.004.3311.3711.00000.1216	N		11
▷	3724146	1GB Memory Chip	119.99	12213 CC_ACCT_LOC_DIV_	AB10.000.1000.0000.00000	N		7

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 renders in the table. For any other value of the popup's `contentDelivery` attribute, the flexfield column in the table will be blank.

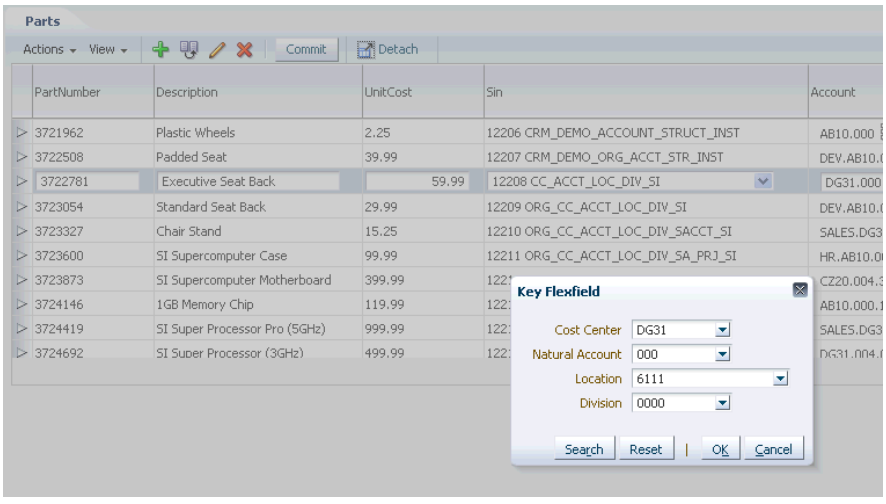
For more information about tables and popups, see the "Using Tables and Trees" and "Using Popup Dialogs, Menus, and Windows" chapters of the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

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 user clicks the icon, instead of the standard popup, a page displays 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 23–13.

Figure 23–13 Example of a Search for a Key Flexfield Code Combination



Note: You do not need to enter values for all segments when searching for a key flexfield code combination.

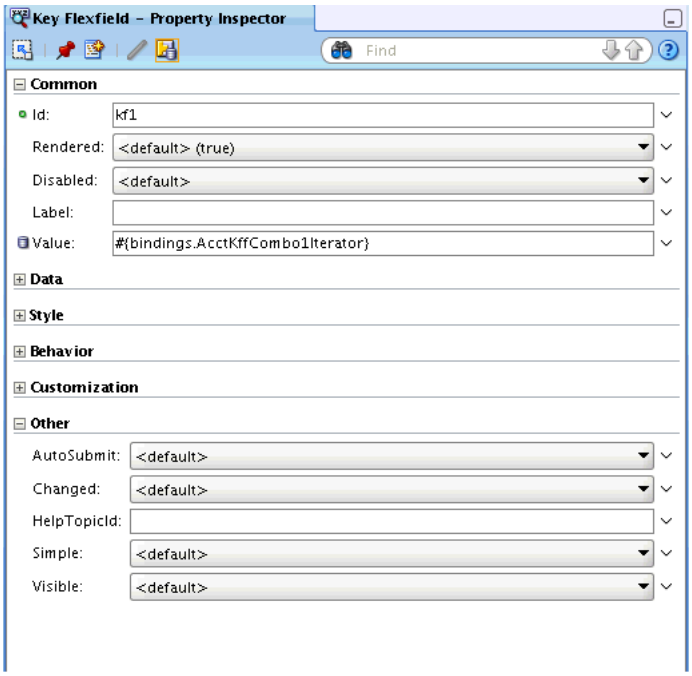
23.3.5 How to Configure Key Flexfield UI Components

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 partial usage level.

23.3.5.1 Configuring 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 23–14.

Figure 23–14 Key Flexfield Property Inspector — Common Tab



The significant properties on the Common, Data, Style, Behavior and Other property tabs are listed in Table 23–6.

Table 23–6 Key Flexfield Properties

Tab > Property	Description
Common > **Id**	The ID of the flexfield.
Common > **Rendered**	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. EL expressions are allowed on ADF Form components and ADF Table components. On ADF Table components, you use expressions to control this property on a row by row basis.

Table 23–6 (Cont.) Key Flexfield Properties

Tab > Property	Description
Common > **Label**	The prompt that should be rendered on the page. Also used for the title of popup components. Note that if the Label property does not have a value, the title of a popup component defaults to `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.
Common > **Value**	The value of the flexfield. This should be an EL pointing to an iterator object. This field is also visible on the Data tab.
Data > **Accessor**	The name of the accessor between the (consumer) application view object and the flexfield view object.
Style > **StyleClass**	The style class of the flexfield.
Style > **InlineStyle**	The inline style of the component.
Style > **Width**	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.
Behavior > **Required**	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 components and ADF Table components. On ADF Table components, you use expressions to control this property on a row by row basis.
Behavior > **ReadOnly**	Indicates whether the key flexfield is rendered as read-only. Values can be `True` or `False` (default). When this property is set to `True`, 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 **Disabled** property. EL expressions are allowed on ADF Form components and ADF Table components. On ADF Table components, you use expressions to control this property on a row by row basis.

Table 23–6 (Cont.) Key Flexfield Properties

Tab > Property	Description
Behavior > **Disabled**	Indicates whether the UI control associated with this key flexfield can be operated. Values can be `True` or `False` (default). When this property is set to `True`, the flexfield segment values and the associated popup or LOV lookup controls are rendered, but are dimmed and cannot be modified or operated. The **ReadOnly** property, when set to `True`, takes precedence over this property. EL expressions are allowed on ADF Form components and ADF Table components. On ADF Table, you use expressions to control this property on a row by row basis. Note that flexfield will be disabled if the current master row for the flexfield does not have a valid SIN value defined.
Behavior > **PartialTriggers**	The IDs of the components that should trigger a partial update in the flexfield (`String[]`).[1] EL expressions are allowed.
Behavior > **ValueChangeListener**	A method reference to a value change listener (`javax.faces.el.MethodBinding`). 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.
Behavior > **Binding**	An EL reference that will store the component instance on a bean (`oracle.apps.fnd.applcore.flex.ui.KFFComp`). Requires an EL expression.
Other > **AutoSubmit**	Indicates whether key flexfield values entered by the 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 application 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 components and ADF Table components. On ADF Table components, you use expressions to control this property on a row by row basis.
Other > **DefaultSIN**	For partial key flexfields only. Defines the default SIN value to use to define the structure when no rows exist.
Other > **Changed**	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 components and ADF Table components. On ADF Table components, you use expressions to control this property on a row by row basis.

Table 23–6 (Cont.) Key Flexfield Properties

Tab > Property	Description
Other > **Simple**	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 components and ADF Table components. On ADF Table components, you use expressions to control this property on a row by row basis.
Other > **Visible**	Indicates whether 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 components and ADF Table components. On ADF Table components, you use expressions to control this property on a row by row basis.

[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. Note — 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.

23.3.5.2 Configuring Label-Based Segment UI Properties

Key flexfields support finer control of segments in the user interface based on their segment labels, using a number of additional properties that you can set in the flexfield XML with literal values or EL expressions. These properties are attributes of the `flexfieldLabeledSegmentHint` element.

Note: This element can be used only to configure the segment UI properties of key flexfield partial 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 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 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 non-required segment to 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 them by inserting customization elements into the UI metadata.

For information about using EL expressions, see the "Creating ADF Data Binding EL Expressions" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

You apply these properties by dragging and dropping the following element from the Component Palette into the key flexfield element, as a child of the `keyFlexfieldpartial` element:

<fnd:flexfieldLabeledSegmentHint *propertyname1*="*value*" [*propertyname2*="*value*" [*propertyname3*="*value*" [*propertyname4*="*value*" [*propertyname5*="*value*" [*propertyname6*="*value*" [*propertyname7*="*value*"]]]]]]>

23.3.5.3 Configuring Partial Usage UI Properties

Key flexfields support finer control of partial usages in the user interface with a number of additional properties that you can set in the flexfield XML with literal values or EL expressions. These properties are attributes of the `keyFlexfieldPartial` element. By using EL expressions at the `.jspx` 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 partial usage:

- **`rendered`** — Indicates whether the flexfield is visible on the application page.

- **`required`** — Indicates whether the flexfield must have a value.
- **`readonly`** — Indicates whether users can modify the flexfield.

The default values of these properties are derived from the flexfield metadata, but you can override them by inserting customization elements into the UI metadata.

> **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 required in the page metadata.

For information about using EL expressions, see the "Creating ADF Data Binding EL Expressions" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

You apply these properties by dragging and dropping the following element from the Component Palette into the key flexfield element:

<fnd:keyFlexfieldPartial *propertyname1*="*value*" [*propertyname2*="*value*" [*propertyname3*="*value*"]]>

23.3.6 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 foreign key view object and its linked key flexfield view object. End 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.

23.3.6.1 Setting 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 foreign key view object, generate the row implementation class for the foreign key 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Before you begin:

1. Create a view link between your application's foreign key view object and the key flexfield polymorphic view object as described in Section 23.3.1, "How to Create Key Flexfield View Links."
2. Add the key flexfield's view instance to the application module for your application as described in Section 23.3.3, "How to Add a Key Flexfield View Object Instance to the Application Module."

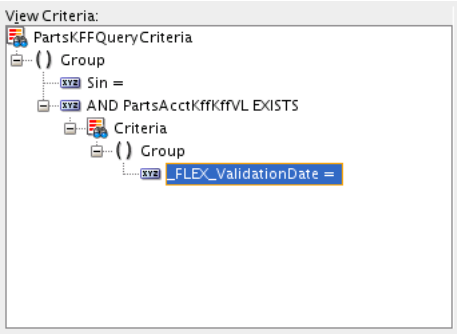
3. Nest the key flexfield application module instance in the application module for your application, as described in Section 23.3.2, "How to Nest the Key Flexfield Application Module Instance in the Application Module."
4. Ensure that the discriminator attribute in the foreign key view object, such 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

To set up the business component model layer:

1. In the Application Navigator, double-click the foreign key view object.
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 foreign key 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 23–15.

Figure 23–15 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 **Java** navigation tab.
16. If the **Java Classes** section does not display a view object class, click the **Edit** icon to generate and configure Java implementation classes, and complete the following steps:
    a. In the Select Java Options dialog, select **Generate View Object Class**.
    b. Optionally select **Include bind variable accessors**, **Include custom java data source methods**, or both.
    c. Click **OK**.
17. In the **Java** navigation tab, click the path shown for the **View Object Class** to open it in the source editor. This is the class that ends with `VOImpl`.
18. In the source editor, override the `getCriteriaItemClause()` method from the `oracle.jbo.ViewCriteriaItem` package as shown in Example 23–10.

    Set `VIEW_CRITERIA_NAME` to the name of the view criteria that you just created, and set `KFF_ACCESSOR_NAME` to the view accessor from the view link between the foreign key view object and the key flexfield polymorphic view object.

    Note: If the foreign key view object contains more than one key flexfield, the `getCriteriaItemClause()` method must call `getCriteriaItemClauseWhenKffExposedinQueryPanel()` for each key flexfield, passing the appropriate criteria name and KFF Accessor Name.

Example 23–10 getCriteriaItemClause() Method

```
private static String VIEW_CRITERIA_NAME="PartsKFFQueryCriteria";
private static String KFF_ACCESSOR_NAME="PartsAcctKffKffVL";
@Override
public String getCriteriaItemClause(ViewCriteriaItem vci) {

  String returnString;
  returnString = null;
  returnString =
    this.getCriteriaItemClauseWhenKffExposedinQueryPanel(
      vci, VIEW_CRITERIA_NAME, KFF_ACCESSOR_NAME);

      return returnString;
}
```

19. Save your changes.

23.3.6.2 Creating the Query Search Form

To create a query search form that contains a key flexfield, you add an ADF Query Panel with Table component to the page and drop the key flexfield into the table. You 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

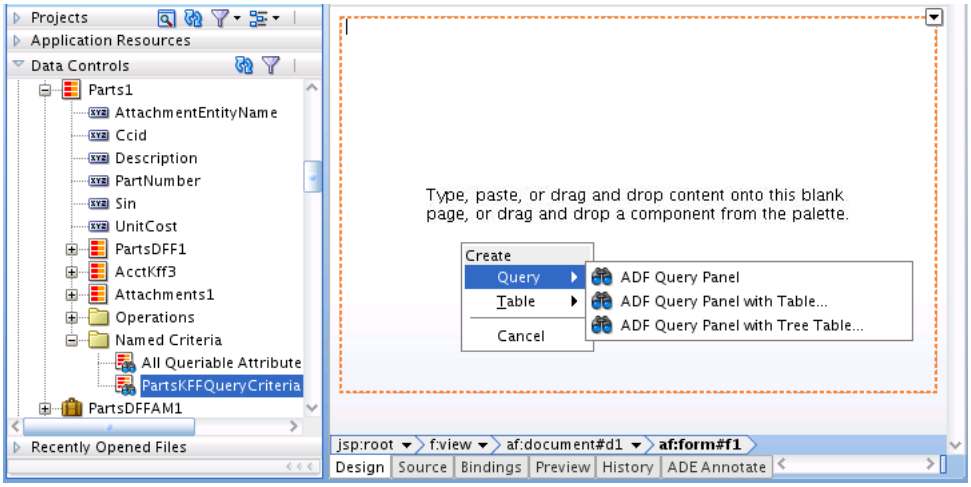
Before you begin:

1. Prepare the business model component as described in Section 23.3.6.1, "Setting Up the Business Component Model Layer."
2. If you are working with partial flexfield segments and the flexfield is an ADF Table component that is wrapped in an Applications Table component, review Section 23.3.4.2, "Ensuring Proper Handling of New Rows" and Section 23.3.4.5, "Dynamically Refreshing Partial Flexfield Segments and Segments on a Combinations Page."

To create the query search form:

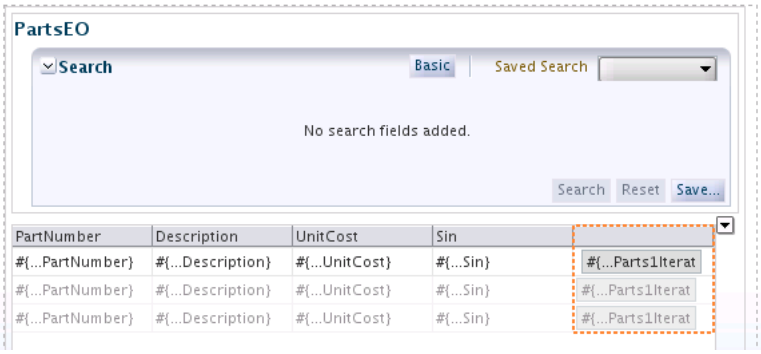
1. Open the JSPX page to which you want to add the search form.
2. From the Data Controls panel, select the foreign key view object's data collection and expand the Named Criteria node to display a list of named view criteria.
3. Drag the view criteria that you created in Section 23.3.6.1, "Setting Up 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 23–16.

Figure 23–16 Query Context Menu



5. In the Edit Table Columns dialog, you can rearrange any column and select table options.
6. 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 Example 23–17.

Figure 23–17 Key Flexfield Column Added to Table



7. In the user interface project, create a custom bean that implements `oracle.adf.view.rich.event.QueryOperationListener` as shown in Example 23–11.

   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 foreign key view object and the key flexfield view object.

Example 23–11 Custom Listener Java Class

```
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_ACCCESSOR_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_ACCCESSOR_NAME)){
                    ViewCriteria nvc = itemimpl.getNestedViewCriteria();

                    ViewCriteriaRow nvcr =
                     (ViewCriteriaRow) nvc.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
   }

}
```

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.

8. 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}`.

23.4 Using 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.

23.4.1 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 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 application 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 launches a key flexfield pop-up window to search for a code combination.
- When the code-combination ID attribute of a foreign key view object 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

23.4.1.1 Creating a View Accessor to Define a Code Combination Constraint

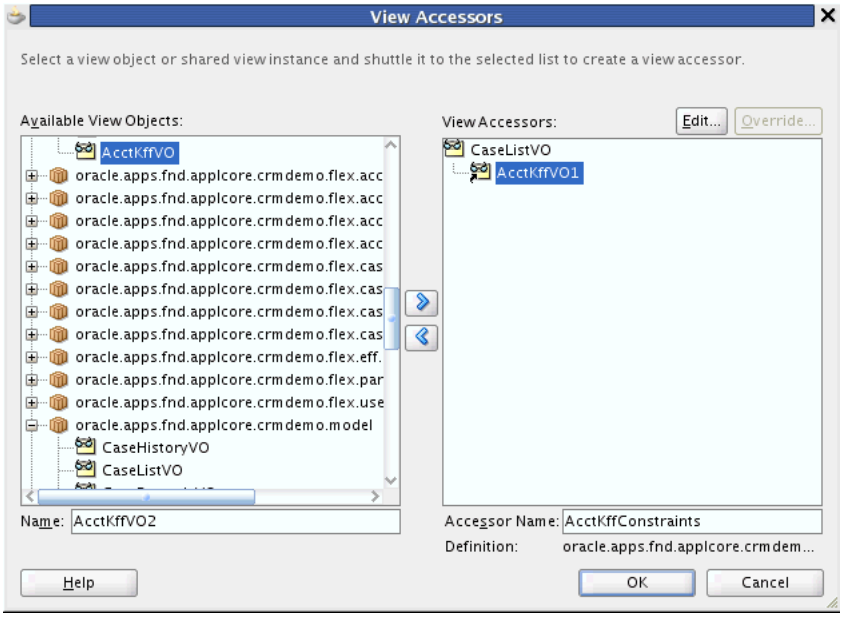
In a key flexfield ADF Business Components model, the code combination constraints are defined in a view object (the foreign key view object) that references the code combinations. Although these constraints are, in a way, validation rules for code combinations, they are not ADF Business Components validators.

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 base key flexfield view object, as shown in Figure 23–18.

Figure 23–18 Code Combination Constraint View Accessor



The destination of the view accessor is the base key flexfield 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`

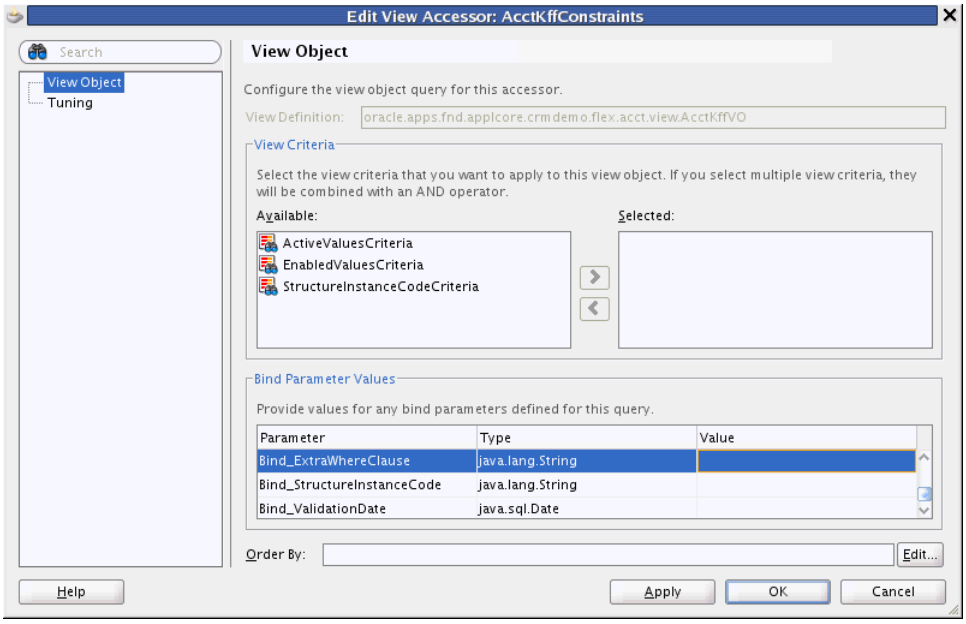
For example, for a view link accessor named `AcctKff`, Figure 23–18 shows the accessor name `AcctKffConstraints`.

4. Edit the view accessor to define the bind parameter values, as shown in Figure 23–19.

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 recreated the business components and overwritten the old ones. You can close the application and remove it from JDeveloper. When you open the application again, JDeveloper should load the latest definitions, including the bind parameters.

Figure 23–19 Code Combination Constraint Bind Parameter Values



There are four types of code combination constraints. You apply a constraint type by providing a value for the appropriate bind parameter for the constraint type. 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 Section 23.4.1.2, "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 Section 23.4.1.3, "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 Section 23.4.1.4, "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 Section 23.4.1.5, "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`

23.4.1.2 Constraining 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 for editing as described in Section 23.4.1, "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 ten pre-defined bind parameters, `BindVar0` to `BindVar9`, use a colon and the bind parameter name; for example, `:BindVar3`.

   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)"
   ```

23.4.1.3 Constraining 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`, 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 don't supply a validation date, the current database date will be used.

To set the Bind_ValidationDate parameter

1. Create the view accessor and open for editing as described in Section 23.4.1, "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 *`yyyy-mm-dd`*.

   In a search user interface, the supplied validation date also affects the list of values of a segment. For example, the 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.

23.4.1.4 Constraining 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 in the loader file 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 partial 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 partial usages.

23.4.1.4.1 How to Create Validation Rules You 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 only apply 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` 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 Section 23.4.1.4.2, "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 23–7 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 23–7 Lexical References

Lexical Type	Lexical Code	Example	Notes
`VALUE`	`VALUE`	`&{VALUE.VALUE}`	Represents the VALUE column in segment lists of values and represents the segment column in combination lists of values.
`VALUE_ATTRIBUTE`	*value attribute code*	`&{VALUE_ATTRIBUTE.GL_ACCOUNT_TYPE}`	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.

For example, if the following where clause was registered as a validation rule for a segment, the derived SQL query to retrieve the segment's list of values would be similar to Example 23–12, and the derived SQL query to retrieve the combination list of values would be similar to Example 23–13.

```
GL_AFF_AWC_API_PKG.gl_valid_flex_values(
  :{FLEXFIELD.VALIDATION_DATE}, &{VALUE.VALUE}) = 'Y')
```

Example 23–12 SQL Query to Retrieve Segment's List of Values

```
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 23–13 SQL Query to Retrieve Combination List of Values

```
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'))
```

> **Tip:** To learn how to access documentation about using the `FND_FLEX_KF_SETUP_APIS` PL/SQL package, see Section 23.2.1.6, "What You May Need to Know About the Key Flexfield Setup API."

23.4.1.4.2 How to Set the Bind_ValidationRules Parameter You 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 for editing as described in Section 23.4.1, "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:

   ```
   VALIDATION_RULE1;VALIDATION_RULE2
   ```

   The validation rules are pre-defined as part of the key flexfield definition. The supplied list is the list of rules that need to be applied when searching for a code combination.

   Note the following caveats 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, 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 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.

23.4.1.5 Enabling or Disabling 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 Section 23.2.4.2, "Enabling Dynamic Combination Insertion".

To set the Bind_DynamicCombinationCreationAllowed parameter

1. Create the view accessor and open for editing as described in Section 23.4.1, "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.

23.4.2 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 access using the flexfield application module or the flexfield view row.

Example 23–14 is an example of Java code that retrieves segment label information from a deployed flexfield using the flexfield application module.

Example 23–14 Retrieving Segment Label Information Using the Flexfield Application Module

```
// 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 need to 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 somehow you 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 23–15 is an example of Java code that retrieves segment label information from a deployed flexfield using the flexfield view row.

Example 23–15 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();
}
```

For more information about segment labels, see Section 23.2.2, "How to Implement Key Flexfield Segment Labels." For more information about the Java API, refer to the Javadoc.

23.4.3 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 (BI) 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 implementer or administrator can select which individual flexfield segments to make available for use with Oracle Business Intelligence.

23.4.3.1 Enabling 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 enable the flexfield and its segments.

You can set the 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 access documentation about using these procedures, see Section 23.2.1.6, "What You May Need to Know About the Key Flexfield Setup API."

You can enable the segments for business intelligence using the Manage Key Flexfields task, which is accessed from the Oracle Fusion Applications Setup and Maintenance work area.

An alternative method to business intelligence–enable a key flexfield and its segments is to set the `BIEnableFlag` to `Y` at both the key flexfield level and the segment level in the key flexfield seed data file (SDF), as shown in Example 23–16.

Example 23–16 BI Enabled Key Flexfield

```
<KeyFlexfield>
    <ApplicationId>0</ApplicationId>
    <KeyFlexfieldCode>KFF1</KeyFlexfieldCode>
    ...
    <TreeStructureCode>FND_FLEX_TEST_TREE_STRUCTURE1</TreeStructureCode>
    <BIEnabledFlag>Y</BIEnabledFlag>
    ...
    <StructureInstance>
      ...
      <SegmentInstance>
        <SegmentCode>L10_ZEROFILL</SegmentCode>
        ...
        <DefaultValue isNull="true"></DefaultValue>
        <BIEnabledFlag>Y</BIEnabledFlag>
        ...
      </SegmentInstance>
      ...
    <StructureInstance>
    ...
  </KeyFlexfield>
```

23.4.3.2 Producing a Business Intelligence–Enabled Flattened Key Flexfield Model

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.

> **Note:** When you make changes to a business intelligence-enabled flexfield, you use the Import Metadata Wizard to import the changes into the Oracle Business Intelligence repository as described in the "Using Incremental Import to Propagate Flex Object Changes" section in the *Oracle Fusion Middleware Metadata Repository Builder's Guide for Oracle Business Intelligence Enterprise Edition (Oracle Fusion Applications Edition)*.

Before you begin:

1. Enable the flexfield and the desired segments for Oracle Business Intelligence as described in Section 23.4.3.2, "Producing a Business Intelligence–Enabled Flattened Key Flexfield Model."
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.

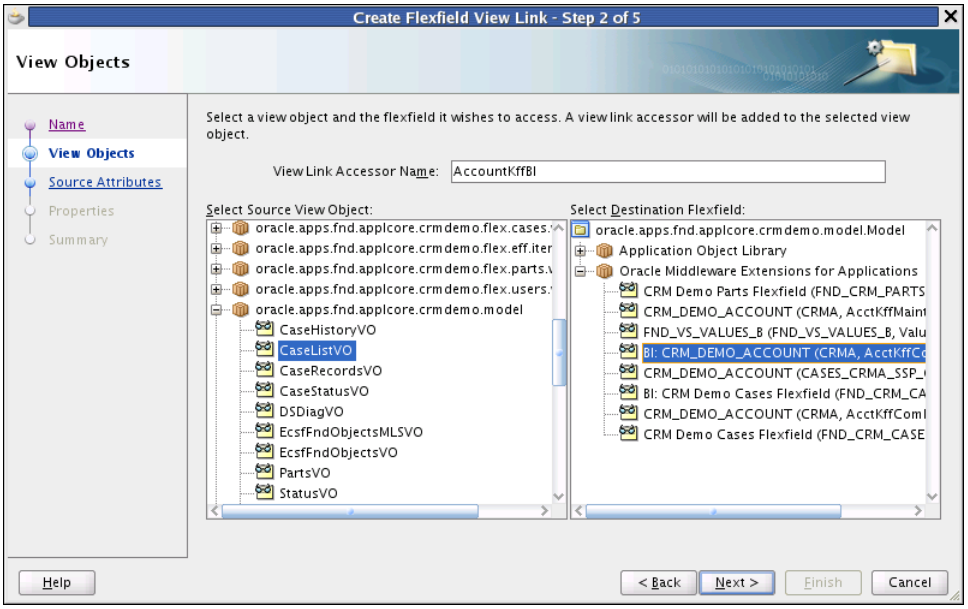
   For more information, see Section 59.8.1, "Designing a Column-Flattened View Object for Oracle Business Intelligence."

To produce a business intelligence–enabled flattened key flexfield model:

1. Create key flexfield business components as described in Section 23.2.4, "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 normal key flexfield view object.
2. Create a view link using the procedure described in Section 23.3.1, "How to Create Key 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 key 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 23–20.

Figure 23–20 Create Flexfield View Link Wizard — View Objects Page



3. Create an application module for use with Oracle Business Intelligence, as described in Section 23.2.4.1.3, "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.

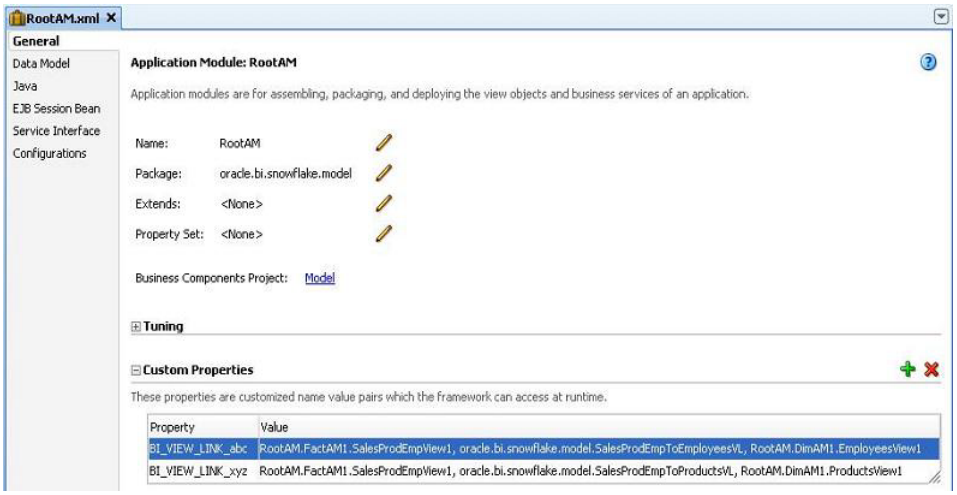
   b. On the Application Modules page of the wizard, add an instance of the key flexfield Oracle Business Intelligence application module as a nested instance of this application module. You can identify the Oracle Business Intelligence application module by the **`analytics`** subpackage under the package root.

   Note: If you already have a product Oracle Business Intelligence application module, you may use it.

4. Following Oracle Business Intelligence Enterprise Edition development guidelines, define the custom properties required to link the master view object instance to the default view instance.

   This is done on the General tab of the nested business intelligence–enabled flexfield application module instance definition, as shown in Figure 23–21.

Figure 23–21 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 normally 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 ADF.

23.4.4 How to Publish Key Flexfield Application Modules as Web Services

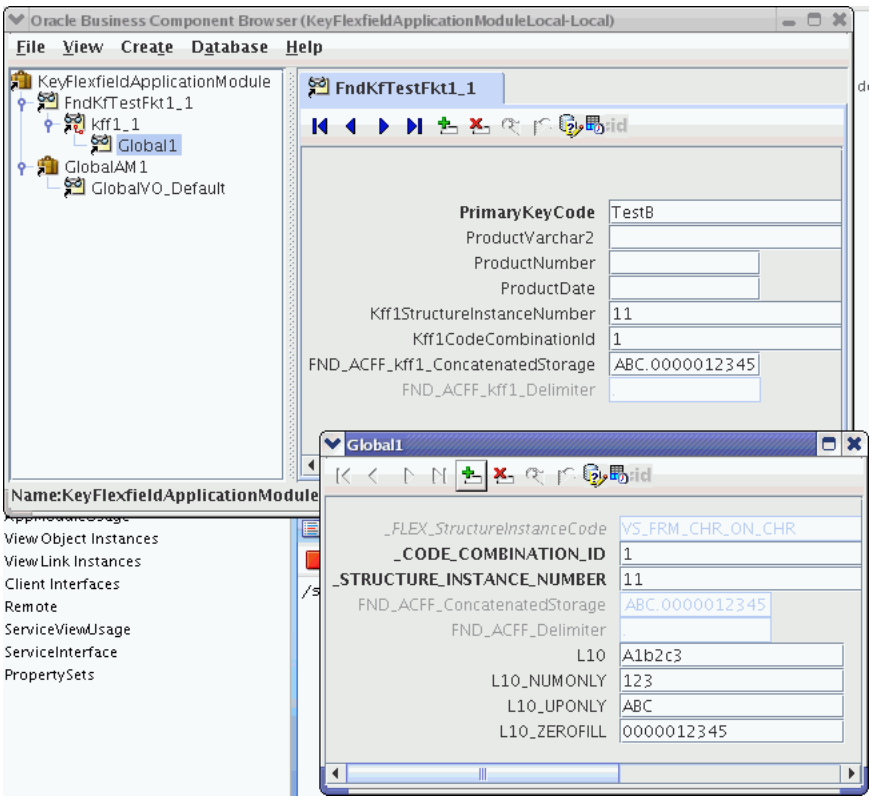
You can make access to a key flexfield available through web services, which will enable you to perform create, read, update, and delete (CRUD) operations on the flexfield data rows. You accomplish this by exposing a key flexfield application module as a web service and adding flexfield service data object support utility methods 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 23–2, 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.

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.

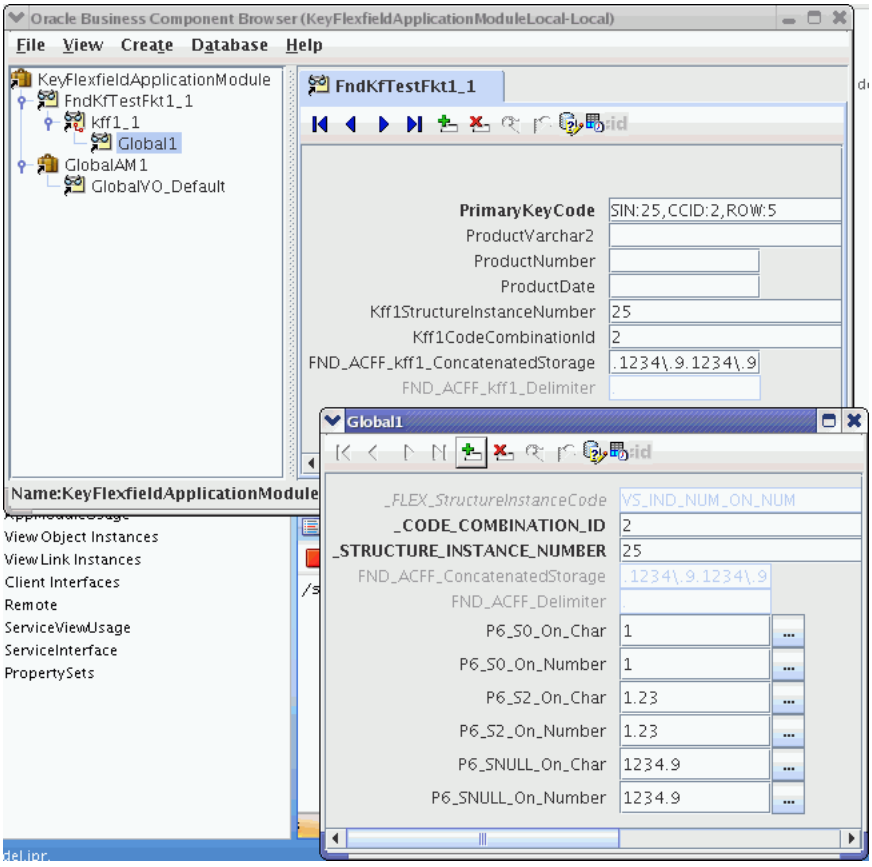
The application module tester view shown in Figure 23–22 corresponds to a particular row in the master view object, displaying the segment structure in the key flexfield with SIN of `11`.

Figure 23–22 Application Module Tester View of Flexfield Row with SIN = 11



The application module tester view shown in Figure 23–23 corresponds to a different row in the master view object, displaying the segment structure in the key flexfield with SIN of `25`.

Figure 23–23 Application Module Tester View of Flexfield Row with SIN = 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.

23.4.4.1 Exposing a Key Flexfield Application Module as a Web Service

You make key flexfield access available through web services by setting a custom property for the flexfield view link, adding a transient attribute to the master view object to store the concatenated flexfield key, service-enabling the master view object, creating the service interface for the project application module within which the key flexfield application module is nested, and adding flexfield service data object support utility methods to the product application module.

> **Note:** In this section, *master view object* refers to the application foreign key view object as illustrated by Figure 23–1.

Before you begin:

1. Create the master entity object and view object over the table that references the key flexfield.
2. Create the flexfield business component as described in Section 23.2.4.3, "Building a Read-Only Reference Model."

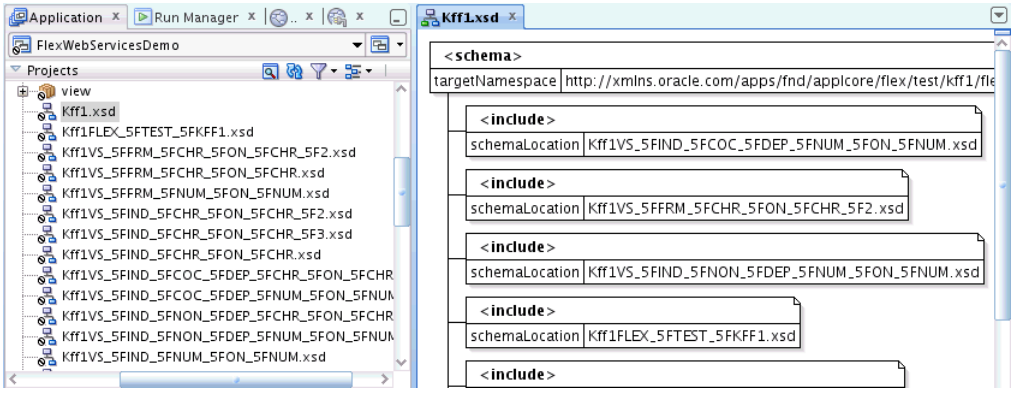
Note: When you generate a flexfield business component, the IDE automatically service enables the business component by generating a Service Data Object (SDO) for the base view object and for every subtype view object.

3. Create a flexfield view link between the master view object and the flexfield business component as described in Section 23.3.1, "How to Create Key Flexfield View Links."
4. Nest the key flexfield application module instance in the project application module as described in Section 23.3.2, "How to Nest the Key Flexfield Application Module Instance in the Application Module."
5. Add the key flexfield view object instance to the application module as described in Section 23.3.3, "How to Add a Key Flexfield View Object Instance to the 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 base flexfield view object and verify that `<include>` elements exist for all the flexfield subtype view objects.

Figure 23–24 Include Elements for the Flexfield Subtype SDOs

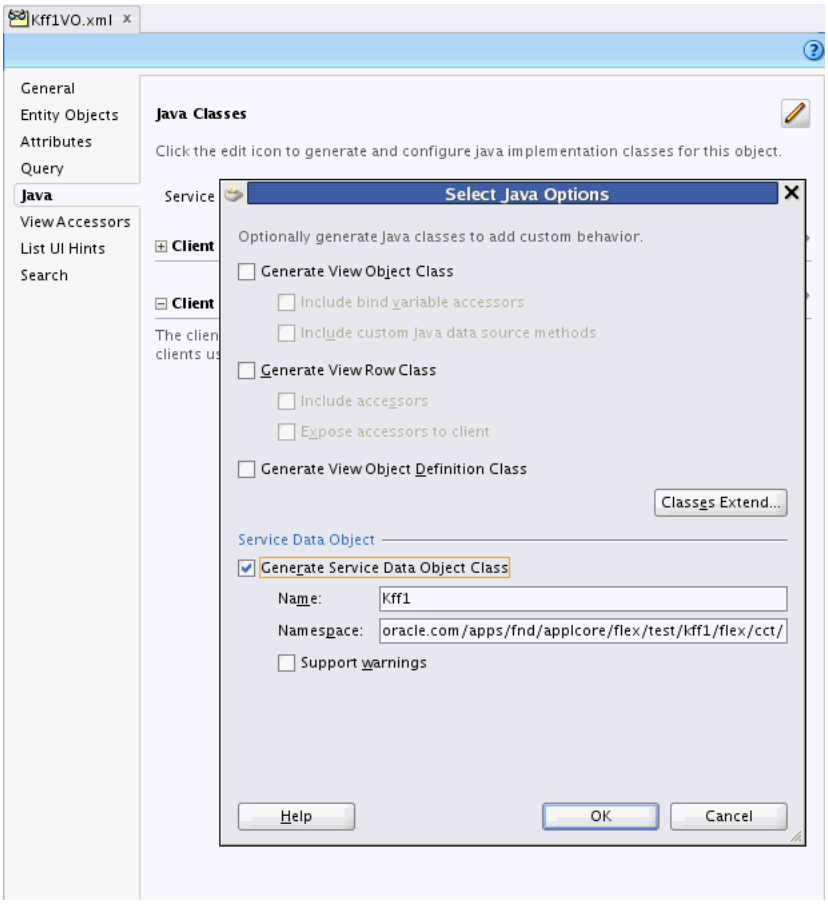


   c. If any subtype view object SDOs are missing, edit the flexfield base view object, and, on the **Java** navigation tab, click the **Edit Java options** icon.

      In the Select Java Options dialog shown in Figure 23–25, 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**.

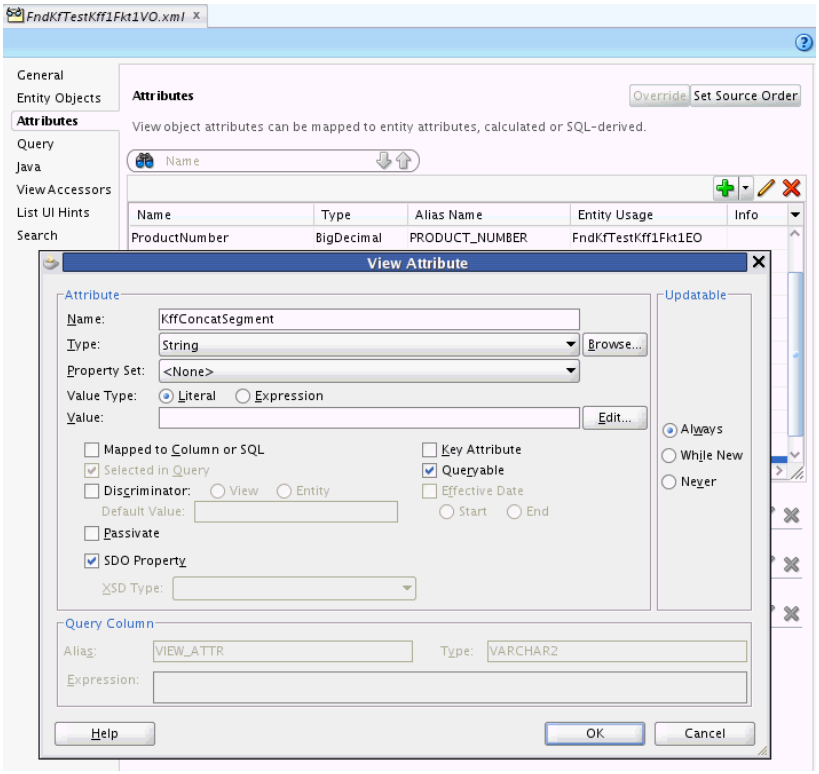
      When the SDO is generated for the base view object of the key flexfield polymorphic view object, generic SDOs are automatically generated for all the base view object subtypes.

Figure 23–25 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 23–26, enter a name for the attribute.

Figure 23–26 Adding a Transient Attribute to the Master View Object



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.

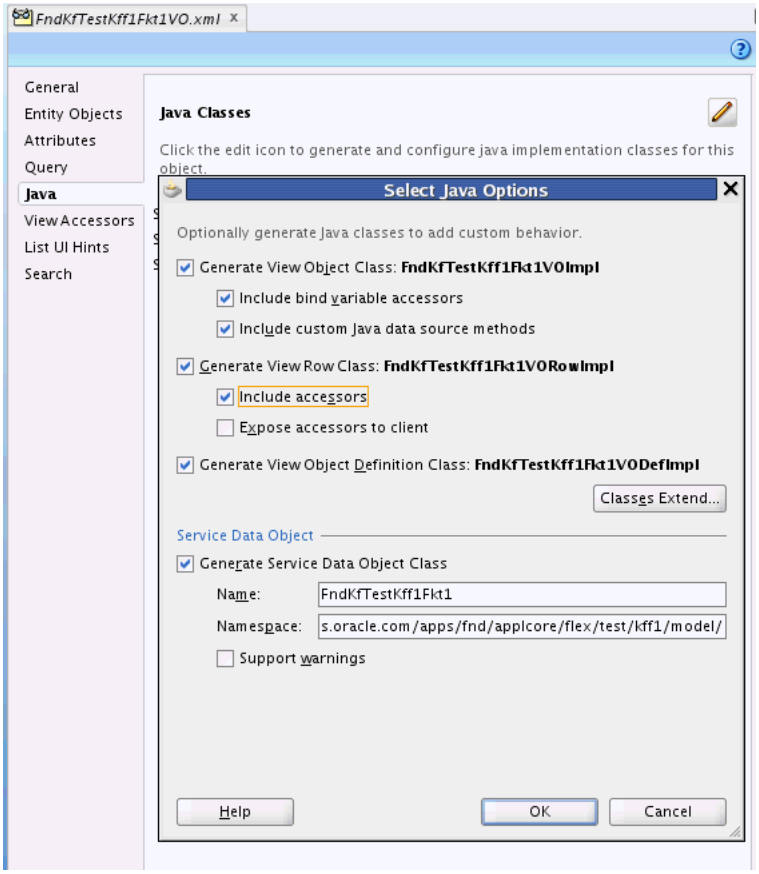
g. Click **OK**.

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 23–27, 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**

Figure 23–27 Generating Java Classes for the Master View Object



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 23–17 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.

Example 23–17 Setter Method for the Transient Attribute

```
  /**
   * 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 acccessor name here";
    String concatAttrName =
      concatAttrPrefix + viewAccessorName + concatAttrPostfix;
    setAttributeInternal(concatAttrName, value);
```

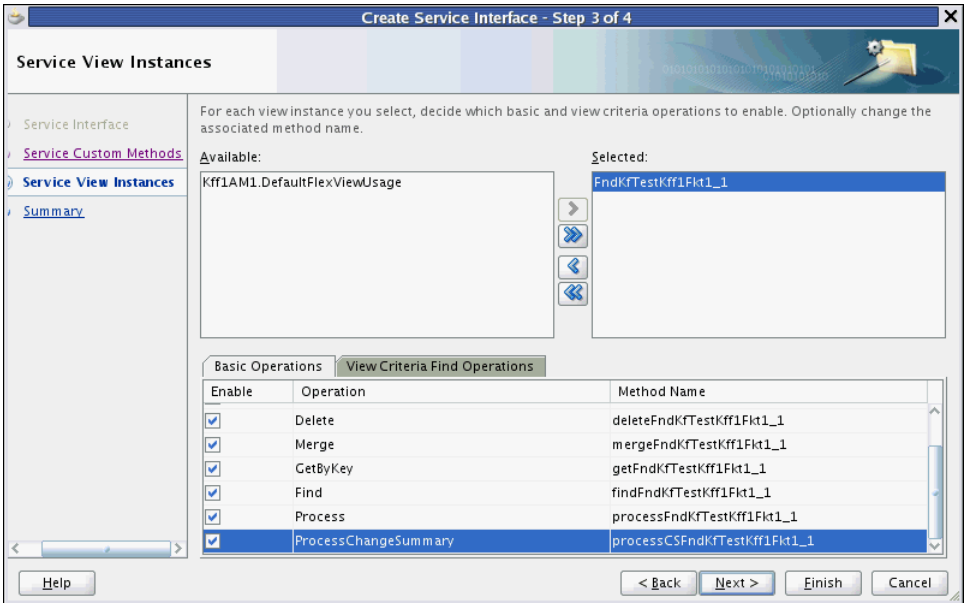
```
    setAttributeInternal(KFFCONCATSEGMENT, value);
  }
```

10. Edit the project application module in which the key flexfield application module instance, master view object instance, and flexfield view object instance are nested.
11. Click the **Service Interface** 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 shuttle 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 shuttle 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 23–28.

    Select the checkbox for each operation of the view object that you want to expose in the service interface and clear the rest.

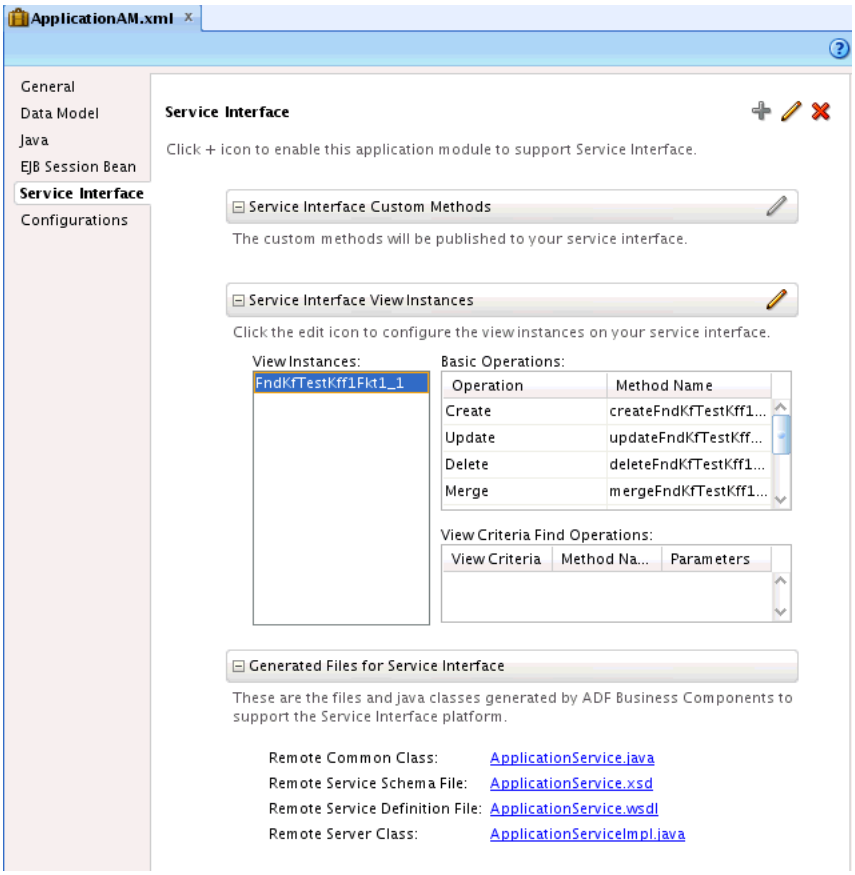
Figure 23–28 Create Service Interface Wizard — Service View Instances Page



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 **Service Interface** 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 23–29.

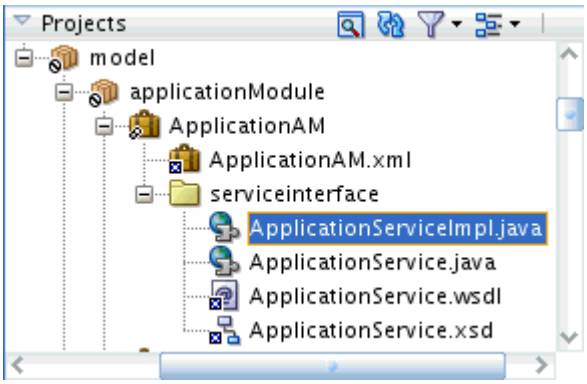
Figure 23–29 Application Module Service Interface Properties



19. Click **Finish**.

    The generated service interface components appear below the application module in the Application Navigator, as shown in Figure 23–30.

Figure 23–30 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 23–18 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 `get`*Flexfield*`SdoSupport` and `get`*Flexfield*`StructureInstanceNumber` methods, replace *getKffMAM1* with the name of the getter method for the nested maintenance application module instance.

Example 23–18 Utility Methods for Flexfield Service Data Object Support

```
// 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 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);
}
```

24. In the overview editor for the product application module, click the **Service Interface** 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, shuttle 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.

23.4.4.2 Testing 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 flexfield service data object support utility methods to test the service.

Before you begin:

1. Make sure that the BC4J Service Client and BC4J Service Runtime libraries are included in your application project.
2. Create a writable maintenance model as described in Section 23.2.4.1, "Building a Writable Maintenance Model."
3. Expose the key flexfield maintenance application module as a web service as described in Section 23.4.4.1, "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 project application module's service (`ApplicationService` in this example).

   This is the service that you created in Section 23.4.4.1, "Exposing a Key Flexfield Application Module as a Web Service" for the project 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 23–19. Modify the host and port number in the `jndiProviderURL` entry to point to your WebLogic Server. The port number is typically `7101`.

Example 23–19 StringRefAddr Elements to Add to Application Module Reference in connections.xml

```
   <Reference
name="{http://xmlns.oracle.com/oracle/apps/fnd/applcore/flex/test/kff1/model/}Appl
icationService"
              className="oracle.jbo.client.svc.Service" xmlns="">
      <Factory className="oracle.jbo.client.svc.ServiceFactory"/>
      <RefAddresses>
         <StringRefAddr addrType="serviceInterfaceName">

<Contents>oracle.apps.fnd.applcore.flex.test.kff1.model.ApplicationService</Conten
ts>
         </StringRefAddr>
         <StringRefAddr addrType="serviceEndpointProvider">
            <Contents>ADFBC</Contents>
         </StringRefAddr>
         <StringRefAddr addrType="jndiName">

<Contents>ApplicationServiceBean#oracle.apps.fnd.applcore.flex.test.kff1.model.App
licationService</Contents>
         </StringRefAddr>
         <StringRefAddr addrType="serviceSchemaName">
            <Contents>ApplicationService.xsd</Contents>
         </StringRefAddr>
         <StringRefAddr addrType="serviceSchemaLocation">
            <Contents>oracle/apps/fnd/applcore/flex/test/kff1/model/</Contents>
         </StringRefAddr>
         <StringRefAddr addrType="jndiFactoryInitial">
            <Contents>weblogic.jndi.WLInitialContextFactory</Contents>
         </StringRefAddr>
         <StringRefAddr addrType="jndiProviderURL">
            <Contents>t3://localhost:port_number</Contents>
         </StringRefAddr>
         <StringRefAddr addrType="jndiSecurityPrincipal">
            <Contents>weblogic</Contents>
         </StringRefAddr>
         <StringRefAddr addrType="jndiSecurityCredentials">
            <Contents>weblogic1</Contents>
         </StringRefAddr>
      </RefAddresses>
   </Reference>
```

4. Run the remote server class for the product application module to deploy the service to the 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 Section 23.4.4.1, "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 23–20 is an example of how a client test program would use the flexfield service data object support utility methods that you added in Section 23.4.4.1, "Exposing a Key Flexfield Application Module as a Web Service."

Example 23–20 Sample Java Code to Test the Web Service

```
public void testSDOAPIs()
{
  // Flexfield service data objects are created automatically based on the
  // flexfield definition. Certain information such as segment codes must be
  // transformed in order 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 (SIC)
  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");

}
```

23.4.5 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 Desktop Integration Developer's Guide for Oracle Application Development Framework. 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.

- Incorporating a key flexfield as a dynamic or static column in an ADF Table component on the page.
- Creating an Excel workbook to integrate with the key flexfield.
- Preparing your Excel workbook to access the column containing the flexfield.

> **Note:** The *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework* does not make explicit reference to technologies documented in this *Oracle Fusion Applications Developer's Guide*, and this guide does not repeat the content in the, *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework* so you must read the *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework* 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 Table component.

  A web page picker popup dialog can be associated with a dynamic column, enabling 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 Table component. 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 Table component needs to expose 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, 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.

23.4.5.1 Configuring ADF Desktop Integration with a Dynamic Column Key Flexfield

When you configure the ADF Table component, 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 *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.
- To make the key flexfield column of the Table component dynamic, set the **DynamicColumn** property in the TableColumn array of the ADF Table component 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 *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.
- 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.

23.4.5.2 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 user of the workbook, changes are not an issue. However, if the 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 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 Table component `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 Table component `DownloadForInsert` method or the `Initialize` method to enable the Table component to reconfigure to accommodate the new flexfield structure.

23.4.5.3 Configuring ADF Desktop Integration with a Static Column Key Flexfield

ADF Desktop Integration supports key flexfields by using tree bindings in an ADF Table component. If you are adding your key flexfield as a static column, you can alternatively use an ADF Read-only Table component. Keep in mind that ADF Read-only Table components support static columns, but not dynamic columns. Popup dialogs support both types.

> **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 *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.

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.

23.4.5.4 Handling Update or Insert of a Key Flexfield Data Row

To handle update or insert of a data row containing a key flexfield in an ADF Desktop Integration table, you call a custom application module method which 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 23–21 and Example 23–22 apply to an ADF Desktop Integration implementation with the key flexfield exposed as a dynamic column. Example 23–23 and Example 23–24 apply to an ADF Desktop Integration implementation with the key flexfield exposed as a static column.

Example 23–21 Updating an Existing Row with a Key Flexfield Dynamic Column

You add this code as an application module method which will be invoked from the **UpdateRowActionId** property of an ADF Table component. This code will be invoked for every row that is updated.

```
Row tempRow = null;  //get KFF child row information based on your KFF View Link
Accessor
KFFViewRowImpl acctRow;
ViewRowImpl kffAcctRow = (KFFViewRowImpl)linerow.getAccountLineKff();

// 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 ADFdi helper class

if (kffAcctRow == null) {
          tempRow = ModelHelper.getAdfdiTempChildRow(linerow, "AccountLineKff");
           kffAcctRow = (ViewRowImpl)tempRow;
       }

Long kffAcctId = null;
String acctSeg=null;

// check whether KFF row is instance of KFFViewRowImpl,
// which means you are updating valid KFF data
// If not that 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 (CREATING NEW SEGMENT COMBINATIONS) FROM ADFdi
Speadhseet
//make sure users supplied 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() return only delimiter information
    //that means user has not supplier 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 23–22 Inserting a New Row with a Key Flexfield Dynamic Column

You add this code as an application module method which will be invoked from the **InsertAfterRowActionId** property of an ADF Table component. This code will be invoked for every row that is inserted.

```
Row tempRow = null;
//Retrieve key flexfield child row information based on your KFF view link
accessor
ViewRowImpl kffAcctRow = (ViewRowImpl)linerow.getAccountLineKff();

//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 ADFdi 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 ADFdi
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 user has not supplier 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));
    }
    }
//setting CCID column with CCID value

     linerow.setDistCodeCombinationId(kffAcctId);
```

Example 23–23 Updating or Inserting a Row with a Key Flexfield Static Column

This code should be added to the setter of the transient attribute in your view object RowImpl.

```
setAttributeInternal(TRANSIENTACCOUNT, value);
//get KFF child row information based on your 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 ADFdi 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 ADFdi
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 23–24 Applying Modified Segment Values to a Cell in a Key Flexfield Static Column

You add this code as a custom application module method which 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)
       //Get temp row from ADFdi
       if (kffAcctRow == null) {
           tempRow = ModelHelper.getAdfdiTempChildRow(headerRow, "AccountKff");
           kffAcctRow = (ViewRowImpl)tempRow;

       }
// Derive and assign value of segments to your transient attribute
// which is created for single cell display in the ADFdi table

headerRow.setTransientAccount(KFFViewRowImpl.getConcatenatedSegments(kffAcctRow));
```

23.5 Completing the Development Tasks for Key Flexfields in Partial 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 *partial usage*.

There are two types of partial usage:

- Single-segment partial 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_COST_CENTER column.

- All-segments partial 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 GL setup table for use in filling in missing values in the subledger account engine.

> **Note:** To incorporate a key flexfield partial usage into your application, you must have already defined and registered the key flexfield master usage on which it is based. See Section 23.2.1.5, "Registering and Defining Key Flexfields Using the Setup APIs," before continuing.

The development tasks for key flexfields in partial mode consist of the following steps:

1. Complete the registration of a key flexfield partial usage (all-segments or single-segment).
2. Create partial mode key flexfield business components based on the partial usage.
3. Create a view link between your application view object and the partial mode key flexfield.
4. The remainder of the development process is essentially the same as the consumer development process for key flexfield master usages. You skip the section on creating key flexfield view links and continue with the tasks described in the following sections:
   a. Section 23.3.3, "How to Add a Key Flexfield View Object Instance to the Application Module"
   b. Section 23.3.2, "How to Nest the Key Flexfield Application Module Instance in the Application Module"
   c. Section 23.3.4, "How to Employ Key Flexfield UI Components on a Page"

   > **Note:** This section contains additional information specific to key flexfield partial usages.

   d. Section 23.3.5, "How to Configure Key Flexfield UI Components"

   > **Note:** This section contains additional information specific to key flexfield partial usages.

23.5.1 How to Register a Key Flexfield All-Segment Partial Usage

All-segment partial usages have a column in the application table for every segment column in the combinations table.

To register an all-segment partial usage:

1. Add columns to your application 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 exactly the same as in the combinations table, with the exception of an optional prefix.

For example, if the column names are `A1` and `A2` in the combinations table, then in the partial 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 PLSLQL registration APIs in the `FND_FLEX_KF_SETUP_APIS` package to register the partial usage.

   To learn how to access documentation about using the APIs, see Section 23.2.1.6, "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 Section 23.2.1.9, "Registering Entity Details Using the Setup APIs."

To implement a key flexfield partial usage, you select the usage at design time. For more information, see Section 23.2.4, "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.

23.5.2 How to Register a Key Flexfield Single-Segment Partial Usage

Single-segment partial usages have one column in the application table to capture a single segment column in the combinations table.

To register a single-segment partial usage:

1. Add the key flexfield segment column that you want to capture to your application 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 Section 23.2.2, "How to Implement Key Flexfield Segment Labels."

3. Use the PLSLQL registration APIs in the `FND_FLEX_KF_SETUP_APIS` package to register the partial usage.

   You must supply the SEGMENT_LABEL_CODE to identify the unique segment label, and COLUMN_NAME to identify the column in your table in which the segment value will be stored.

   To learn how to access documentation about using the APIs, see Section 23.2.1.6, "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 Section 23.2.1.9, "Registering Entity Details Using the Setup APIs."

To implement a key flexfield partial usage, you select the usage at design time. For more information, see Section 23.2.4, "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.

23.5.3 How to Create Partial Mode Key Flexfield Business Components

Zero or more partial usages can be defined for a given flexfield, each one potentially on a different application table.

Before you begin:

One or more required libraries might have not been automatically included in your application 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 standard wizard, create application entity objects based on the combinations tables you have defined. Make sure of the following:

- At least one customization class is included in `adf-config.xml`.

  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 *Oracle Fusion Applications Extensibility Guide*.

- These entity objects are directly modeled on the combinations tables; hence they contain the fixed (non-flexfield) columns, if any, along with all of the flexfield columns. In general, all columns should be included.
- The entity objects have primary keys defined.
- 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.

- 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 flexfield usage to which it will provide data, as described in Section 23.2.1.9, "Registering Entity Details Using the Setup APIs."

To create partial mode key 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. In the New Gallery, navigate to **Business Tier** > **ADF Business Components** and select **Flexfield Business Components**.
3. Click **OK** to access the Create Flexfield Business Components wizard.
4. On the Role page, select the role that you are taking as you create the flexfield business components:
   - **Developer** — select this role if you are incorporating the flexfield into an application. The business components must be stored in one of your projects. Select the desired project location from the **Project Source Path** dropdown list.

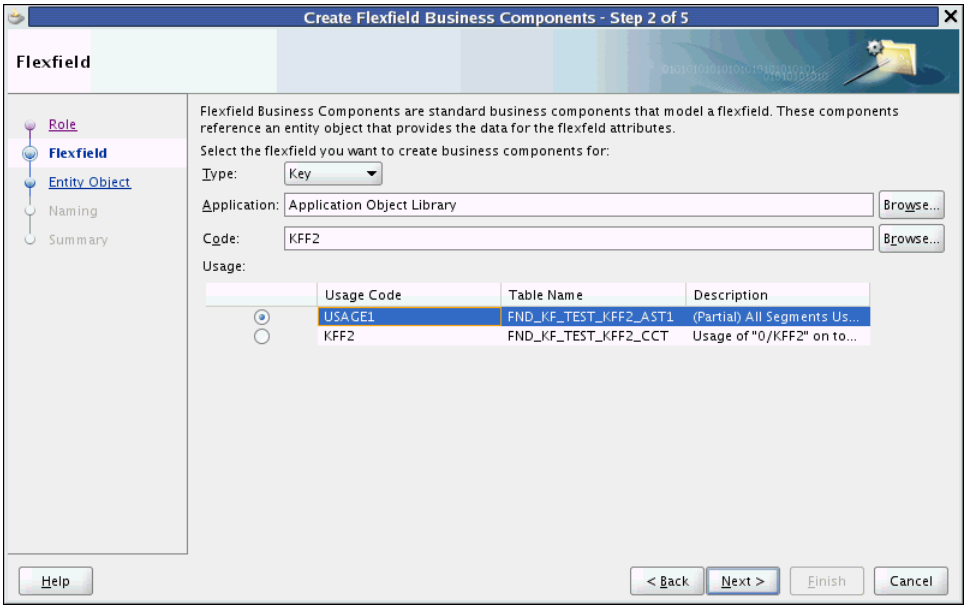
- **Tester** — select this role if you are planning to test your flexfield or test a shared flexfield. In the **Output Directory** field, specify the path of your desired location for the generated business components.

  For more information about testing flexfields, see Chapter 25, "Testing and Deploying Flexfields." For more information about sharing and importing shared flexfields, see Section 23.2.5, "How to Share Key Flexfield Business Components."

  Note: This is not a role in the security sense. It exists only during this procedure, for the purpose of specifying where your generated flexfield business components should be stored.

5. Click **Next**. The Flexfield page appears, as shown in Figure 23–31.

Figure 23–31 Create Flexfield Business Components Wizard — Flexfield Page



6. From the **Type** dropdown list, select **Key**.
7. 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**.
8. In the **Code** field, specify the code of the key flexfield you want to use.

   You can browse for and filter by **Code**.
9. In the **Usage** section, select the table row that contains your desired partial key flexfield usage. Key flexfield usage can be one of the following types:

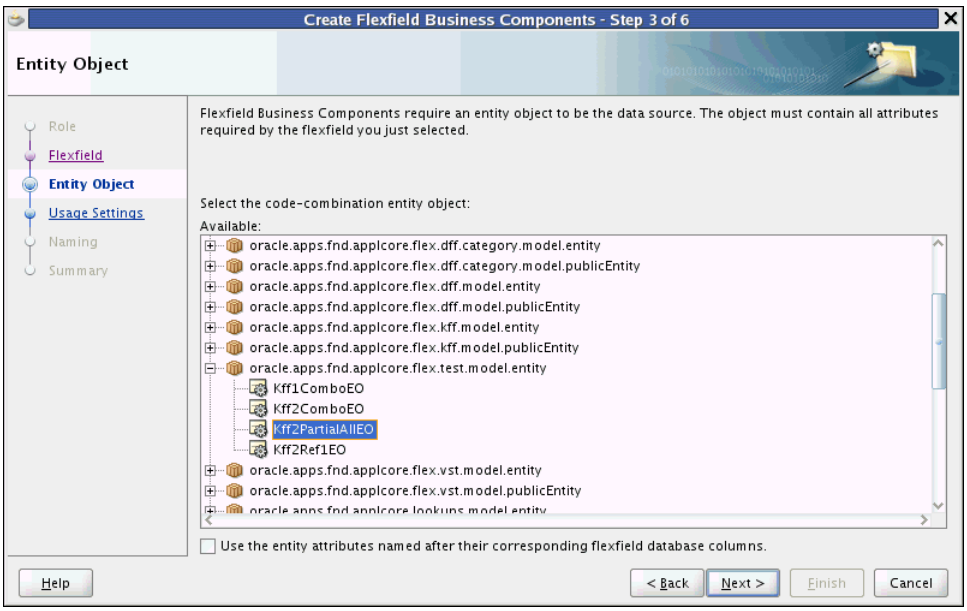
   - An all-segment partial usage of the key flexfield on an application table other than the combinations table. Zero or more partial usages can be defined for a given flexfield, each one potentially on a different application table. You can identify partial usages by the presence of the prefix `(Partial)` in the **Description** field.

- A single-segment partial usage of the key flexfield on an application table other than the combinations table. Zero or more single-segment partial usages can be defined for a given flexfield, each one potentially on a different application table. You can identify single-segment partial 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 partial usages, see Section 23.5.1, "How to Register a Key Flexfield All-Segment Partial Usage."

10. Click **Next**. The Entity Object page appears, as shown in Figure 23–32.

Figure 23–32 Create Flexfield Business Components Wizard — Entity Object Page



11. Expand the tree of available models and select an entity object to use as the data source for the key flexfield.

    Because you selected a partial 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 partial 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.

    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 upper case.

12. 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, select the checkbox labeled **Use the entity attributes named after their corresponding flexfield database columns**. This checkbox is unselected by default.

    When a key flexfield entity object attribute is transient, there is no matching underlying column name. When you select this checkbox, 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. Make sure 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 base table usage as described in Section 23.2.1.9, "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 base table 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 Section 23.5.1, "How to Register a Key Flexfield All-Segment Partial Usage."

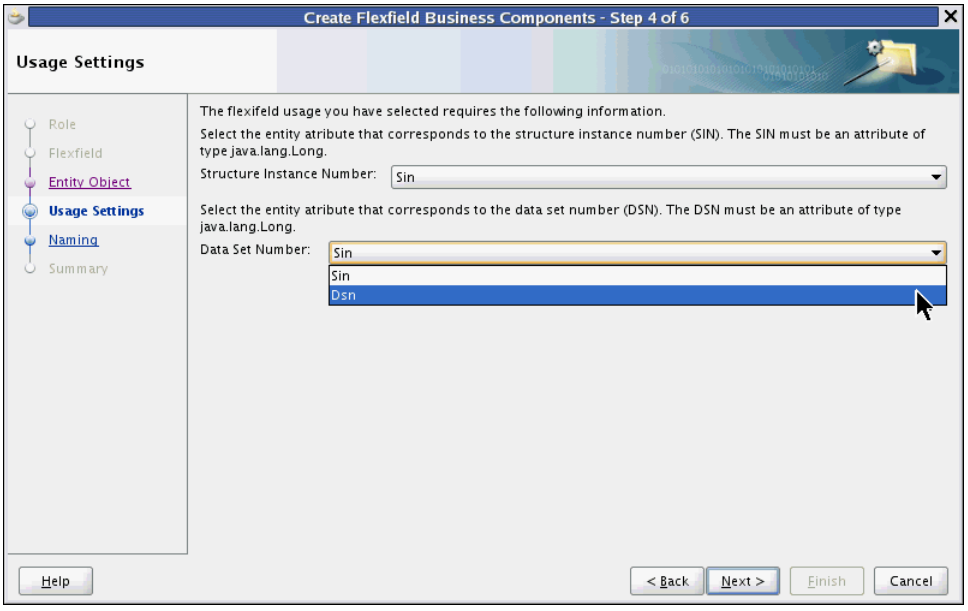
---

13. Click **Next**. The Usage Settings page appears.

    This page contains a **Structure Instance Number** dropdown list, as shown in Figure 23–33. From the dropdown list, select the entity attribute that corresponds to the key flexfield SIN for the partial 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 partial usage. The DSN must be an attribute of type `java.lang.Long`.

Figure 23–33 Create Flexfield Business Components Wizard — Usage Settings Page



14. Click **Next**. The Naming page appears.

To create business components for the key flexfield partial usage that you previously selected, the package name and the object name prefix for the selected entity object must first be registered with that flexfield usage. Text on the Naming page indicates whether this is the case:

- If the selected entity object is registered with the flexfield usage, the Naming page displays the package name and the object name prefix for the entity object. Click **Next** and continue to Step 15.
- If the selected entity object is not registered (as an ADF Business Components usage) with the flexfield usage, the Naming page displays a message to that effect. Take one of the following actions:
  - Click **Back** to return to the Entity Object page and select an entity object that has been properly registered.
  - Click **Cancel** to exit this wizard and register the entity object that you want to use, as described in Section 23.2.1.9, "Registering Entity Details Using the Setup APIs."

15. 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 application 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.

16. Refresh the project to see the newly created flexfield business components in the Application Navigator.

23.5.4 How to Create Partial Mode Key Flexfield View Links

A view link is needed whenever an application view object references your key flexfield. The base view object can have many incoming view links from various application view objects, as a key flexfield is usually shared by many application tables.

Before you begin:

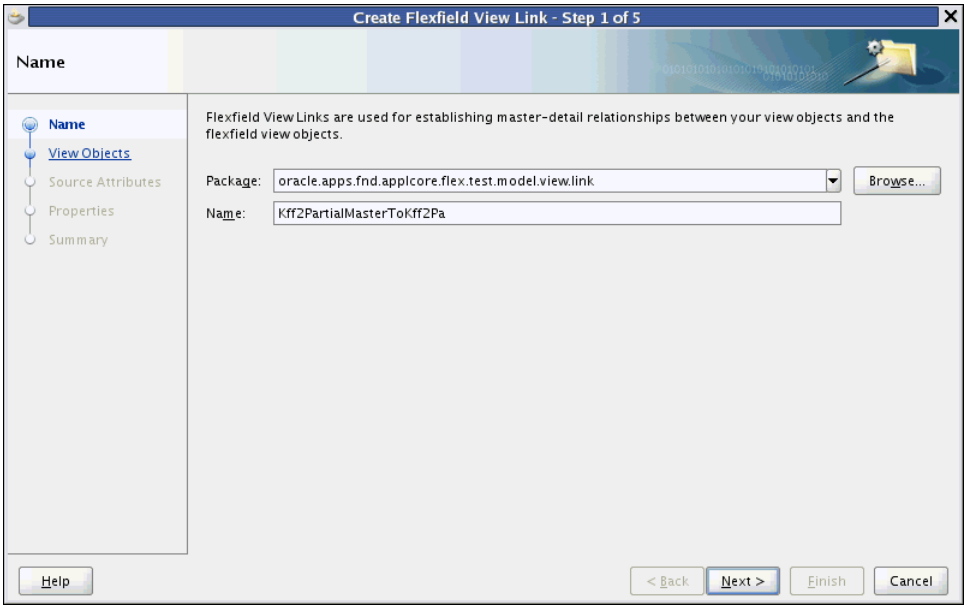
You should have already created a master view object over your entity object using the standard wizard.

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**, **SIN**, and, if present, **DSN**. Ensure that the CCID attribute's **Display** control hint is set to `Hide`.

To create a partial mode key flexfield view link:

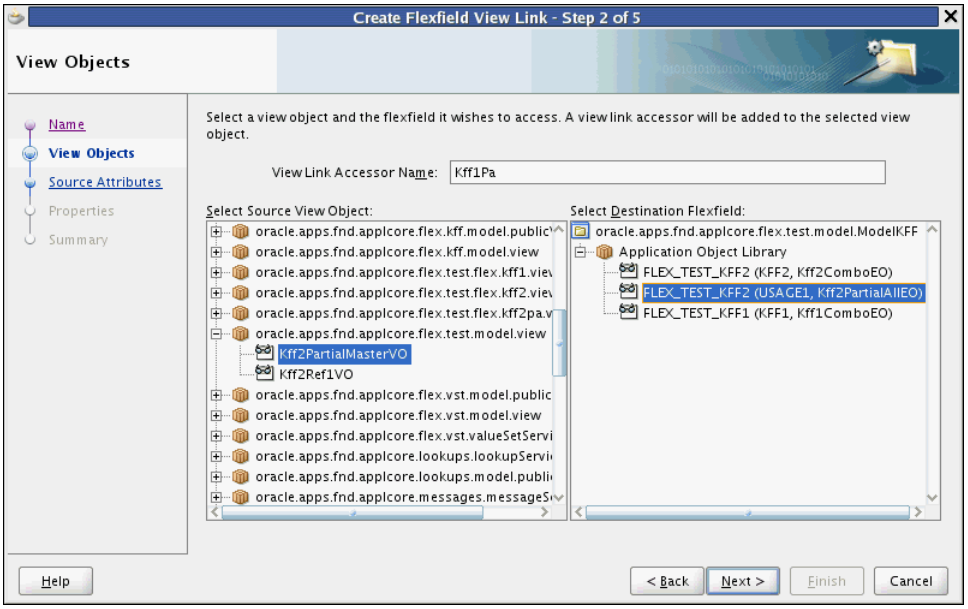
1. In the New Gallery, navigate to **Business Tier** > **ADF Business Components** and select **Flexfield View Link**.
2. Click **OK** to access the Flexfield View Link wizard, as shown in Figure 23–34.

Figure 23–34 Create Flexfield View Link Wizard — Name Page



3. On the Name page, from the **Package** dropdown list, specify a package for the view link.
4. In the **Name** field, enter a name for the partial mode view link.
5. Click **Next**. The View Objects page appears, as shown in Figure 23–35.

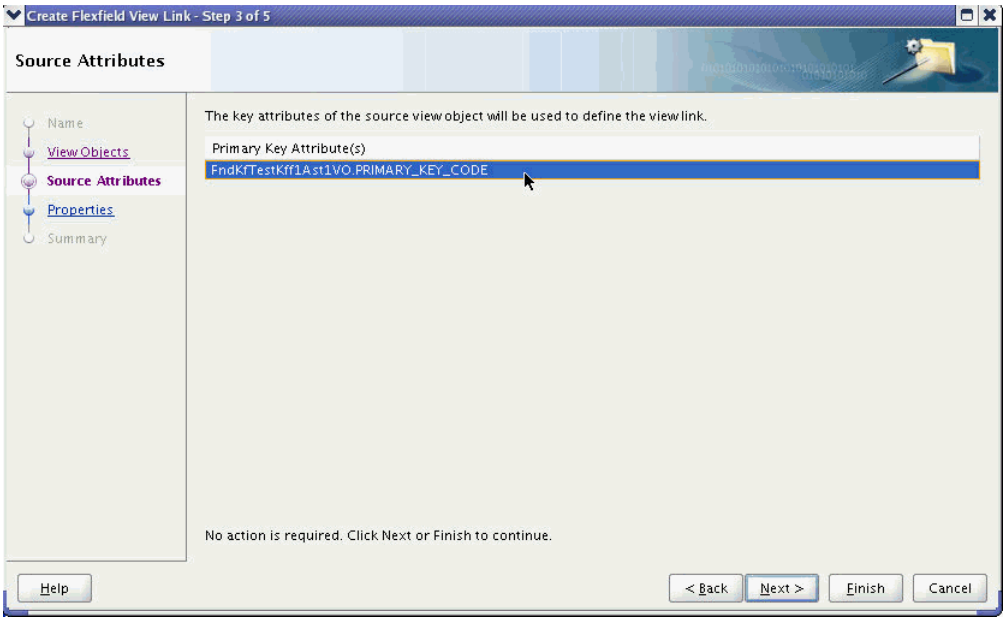
Figure 23–35 Create Flexfield View Link Wizard — View Objects Page



6. In the **Select Source View Object** tree, expand the available partial mode key flexfield view objects from your current project and select a source partial mode view object.
7. In the **Select Destination Flexfield** tree, expand the available flexfield view objects from your project and select a destination base key flexfield 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 23–36.

Figure 23–36 Create Flexfield View Link Wizard — Source Attributes Page for Partial Mode Key Flexfields



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 partial mode key flexfield view link is generated.

23.6 Working with Key Flexfield Combination Filters

A key flexfield combination filter is a set of query criteria that can be applied to a combinations table to specify a subset of code combinations. Once you incorporate a key flexfield 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 23–8:

Table 23–8 Example Combinations Table

SIN	CCID	Summary_Flag	Enabled_Flag	Segment1_Varchar2	Segment2_Varchar2	..
11	77	N	Y	8.5X12	YEL	..
11	78	N	Y	8.5x14	GRN	..

Table 23–8 (Cont.) Example Combinations Table

SIN	CCID	Summary_Flag	Enabled_Flag	Segment1_Varchar2	Segment2_Varchar2	..
14	2	N	Y	ERGO	NYLON	..
14	3	N	Y	HGBAK	LEATHER	..

You could define a filter with the following conditional logic:

```
SIN=14 and Segment1_Varchar2='ERGO'
```

When you apply this filter condition to the combinations table, the result listed in Table 23–9 is presented:

Table 23–9 Example Filter Result

SIN	CCID	Summary_Flag	Enabled_Flag	Segment1_Varchar2	Segment2_Varchar2	..
14	2	N	Y	ERGO	NYLON	..

Key flexfield combination filter conditions are stored in an XMLType database column.

There are three types of combination filters that you can use in your application — standard combination filters, combination filters for Oracle Business Intelligence (BI) Publisher reports, and cross-validation filters.

Standard Combination Filters

With standard combination filters, you determine which key flexfields your end users should be able to filter, and define a dedicated 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 columns.

The filter condition is stored in the database column as XML. At runtime, the filter condition in the XML is converted to a `ViewCriteria` object and applied over the appropriate base key flexfield 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 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 combination filter accessible to application implementers or administrators, you add a key flexfield 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 implementers or administrators will be responsible for populating the table with filter criteria using a provided utility.

Key flexfield combination filters 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/applcore/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 XMLType filter 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 Section 23.6.5, "How to Apply Combination Filters Using the PL/SQL Filter APIs."

Combination filters are removed from an application by removing their accessors.

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 key flexfield combination filter criteria, which are then translated into SQL for inclusion in the report. You accomplish this by creating a flexfield filter view object over the public entity object `FndKfEssFiltersPEO` to access a provided common filter repository table, then 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 (ESS) launches 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 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 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.

Cross-Validation Filters

Cross validation rules leverage the code combination filter infrastructure to apply a pair of filters to new code combinations that are proposed for a key flexfield by administrators or end users, when you have enabled them 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 comprise 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 input.

> **Note:** Cross-validation rule criteria should generally be created and modified only by application implementers 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 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 Section 23.2.3, "How to Implement Cross Validation Rules and Custom Validation."

23.6.1 How to Prepare the Database for Standard Key Flexfield Combination Filters

An XMLType column is required to store filter data in your database. For standard combination filters, you must define an XMLType column for the filter data in your database before you can associate combination filters with key flexfields in your application.

> **Note:** If you are implementing combination filters to support cross validation rules, the required XMLType columns already exist in the `FND_KF_CROSS_VAL_RULES` repository table. If you are implementing 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 key flexfield 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:

   ```
   Create table FND_MYFILTER_KFF1 (ID Number primary key, Info Varchar2(1000));
   ```

2. Use the following Alter script to add a filter column (for example, `Filter`) of type XMLType to your table:

   ```
   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 column must be configured as nullable.
>
> This script is necessary because XDF does not support the XMLType data type.

23.6.2 How to Add Combination Filters to Your Application

To add combination filters to your application, you complete the following tasks:

1. For standard filters only, create an entity object over the table containing the filter column.
2. Create a view object over the filter entity object.
3. Associate the combination filters with key flexfields.
4. Configure, deploy, and test the combination filters.

23.6.2.1 Creating a Filter Entity Object for a Standard Filter

For standard filters, you must create a filter-specific entity object over the table containing the filter 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 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.FndKfEssFiltersPEO`, which became available when you added the Applications Core library to your data model project.
- If you are implementing combination filters to support cross validation rules, use the provided configured entity object:

  `oracle.apps.fnd.applcore.flex.kff.model.entity.KeyFlexfieldCrossValidationRuleEO`

Before you begin:

1. Define a database column of XMLType to store the filter as described in Section 23.6.1, "How to Prepare the Database for Standard Key Flexfield Combination Filters."
2. To use key flexfield 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`) over the table containing your filter 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`.

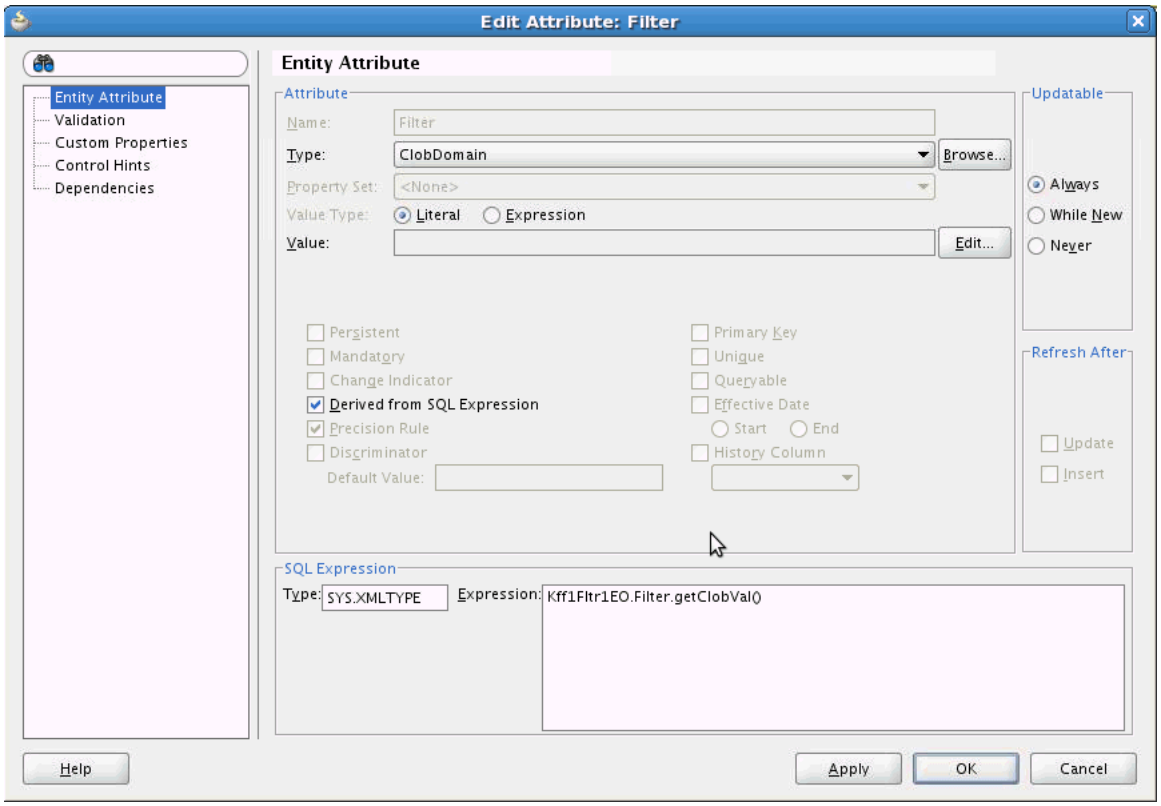
   This property enables the base classes (`OAEntityImpl`) to recognize that the entity object contains a filter attribute.
4. Because the column type of the filter attribute is XMLType, which is not natively supported by ADF Business Components, you must edit the attribute to make it a this a transient 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 23–37.
   a. Specify the **ClobDomain** type.
   b. Select **Derived from SQL Expression**.
   c. Enter an **Expression** such as the following expression:

      ```
      Kff1Fltr1EO.filter.getClobVal()
      ```

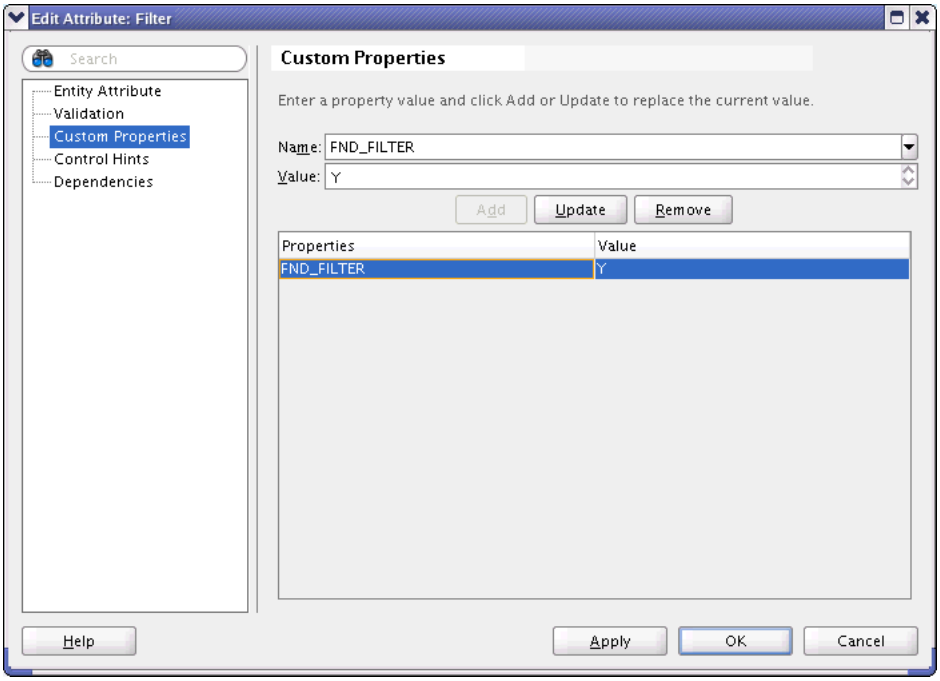
Note: `GetClobVal()` is needed to manage the XMLType column in the database because ADF Business Components currently does not support the XMLType data type natively.

Figure 23–37 Edit Filter Attribute — Entity Attribute Page



6. Click the **Custom Properties** node, as shown in Figure 23–38.

Figure 23–38 Edit Filter Attribute — Custom Properties Page



7. Add the custom filter properties that are listed in Table 23–10 to the filter attribute.

Table 23–10 Custom Filter Properties

Name	Value	Description
`FND_FILTER`	`Y`	A `Y` value indicates that the entity attribute is a filter attribute.
`FND_FILTER_TABLE`	*table-name*	Indicates the name of the underlying table on which this filter attribute is based.
`FND_FILTER_COL`	*column-name*	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.

`FND_FILTER_TABLE_COL_PK`*n*	*primary-key-column-id*	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`

Table 23–10 (Cont.) Custom Filter Properties

Name	Value	Description
FND_FILTER_TABLE_ATTR_PK*n*	*view-object-attribute-name*	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=ID1 FND_FILTER_TABLE_ATTR_PK2=ID2

23.6.2.2 Creating a Filter View Object

You need to create a filter view object (for example, Kff1Fltr1VO) over the filter entity object.

- If you are implementing a standard combination filter, create the view object over the entity object that you created Section 23.6.2.1, "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, create the view object over the public entity object oracle.apps.fnd.applcore.flex.kff.model.publicEntity.FndKfEssFiltersPEO.
- If you are implementing the filter to support cross validation rules, create the view object over 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.

23.6.2.3 Associating Combination Filters with Key Flexfields

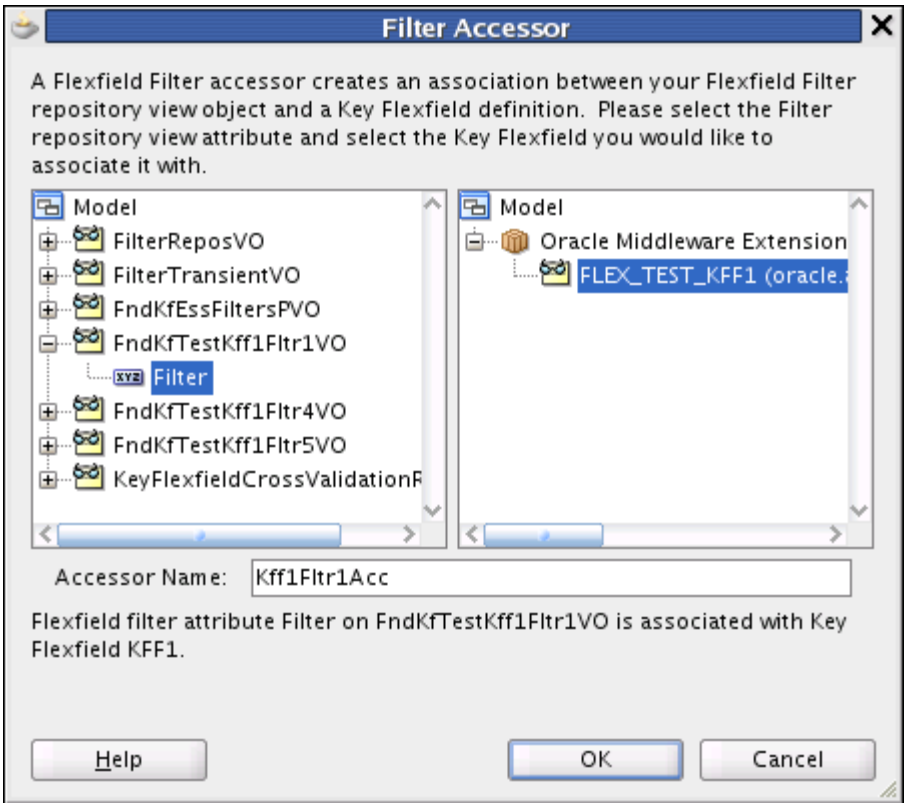
You use the Create Flexfield Filter wizard to create a view accessor from the filter view object's combination filter attribute to a key flexfield view object definition.

> **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 combination filter with a key flexfield:

1. In the New Gallery, navigate to **Business Tier** > **ADF Business Components** and select **Flexfield Filter**.
2. 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 23–39.

Figure 23–39 Filter Accessor Dialog



3. Expand the available flexfields in your current project on the right-hand list and select a key flexfield to be filtered.
4. Enter a name for the filter accessor (with no spaces), then click **OK**.

23.6.2.4 Configuring, Deploying and Testing 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 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 associates with this filter attribute.
5. Create an application module for the filter. In the Data Model page, shuttle the filter view object to the **Data Model** list. In the Application Modules page, shuttle 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 application module tester to make sure that all attributes appear.

23.6.3 How to Employ Combination Filters on an Application Page

You need to 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.

23.6.3.1 Adding Your Key Flexfield Filter to an Application Page

You add a key flexfield filter to an application page by dropping the filter view object on 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.

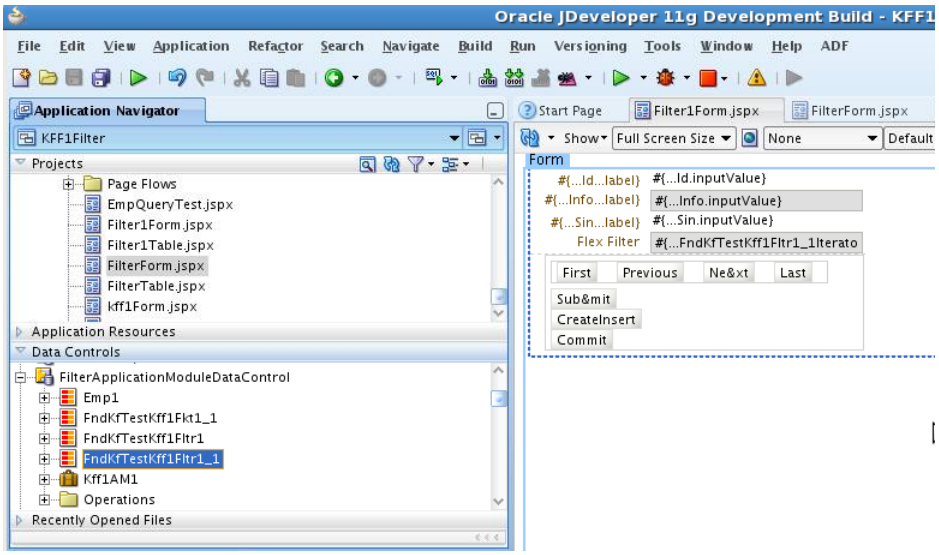
Before you begin:

- Ensure that the Applications Core (ViewController) tag library has been added to the user interface project, as described in Section 3.4, "Adding the Applications Core Tag Library to Your User Interface Project."
- Create the view object, view accessor, and application module for the combination filter as described in Section 23.6.2, "How to Add 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 component or an ADF Table component. Figure 23–40 shows the filter view object dropped onto the page as a form.

Figure 23–40 Filter dropped Onto a Page as an ADF Form Component



3. Make sure the **CreateInsert** and **Commit** actions are included on the page.

> **Note:** These actions enable extemporaneous 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 component, 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 component, 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 23–41 and Figure 23–42. If you want restrict the filter to match on all conditions only, add the **RestrictConjunctionToAND** property and set it to **true**, as shown in Example 23–25 and Example 23–26

Example 23–25 Modified Form-Based Filter Code

```
<fnd:keyFlexFilter value="#{bindings.Kff1Fltr1_1Iterator}"
    accessor="kff1"
    label="#{bindings.Filter.hints.label}"
    id="kff1"
    restrictConjunctionToAND="true"/>
```

Example 23–26 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>
```

When this property is set to **true**, the Filter dialog box does not display the Match options, as shown in Figure 23–43, and the conditions are automatically joined with an AND conjunction.

Figure 23–41 Form-Based Combination Filter User Interface

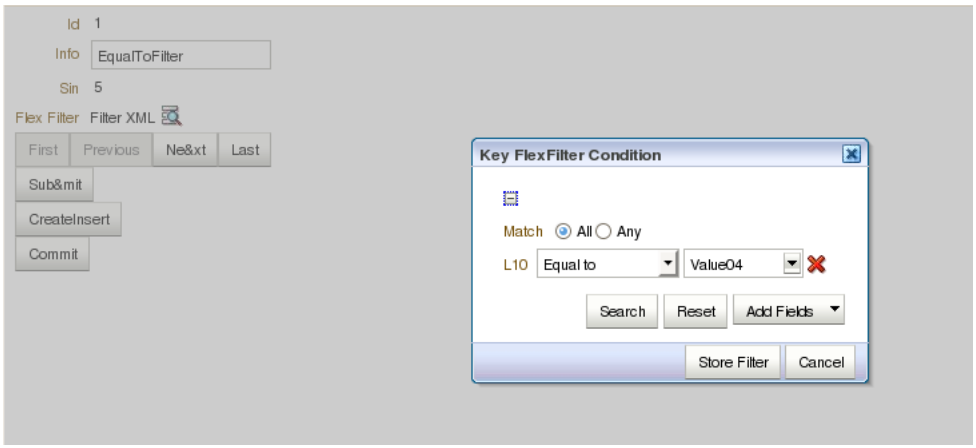


Figure 23–42 Table-Based Combination Filter User Interface

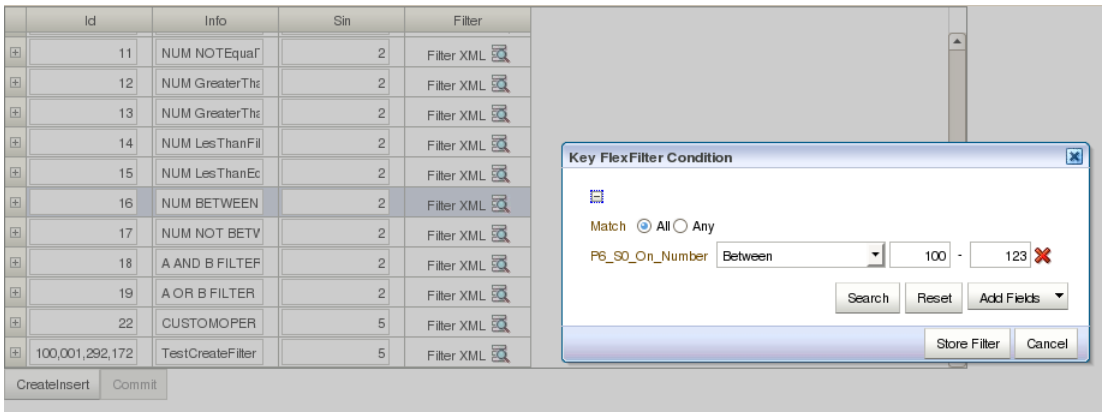
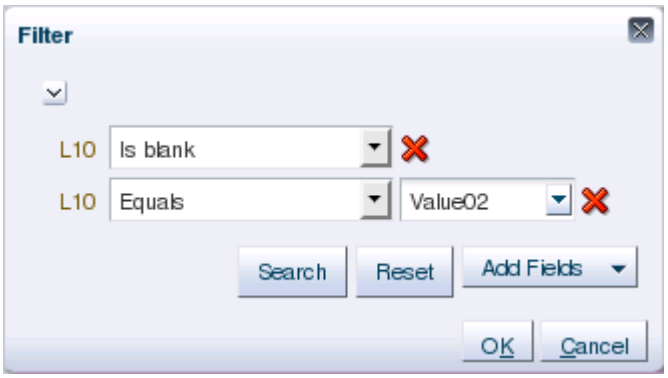


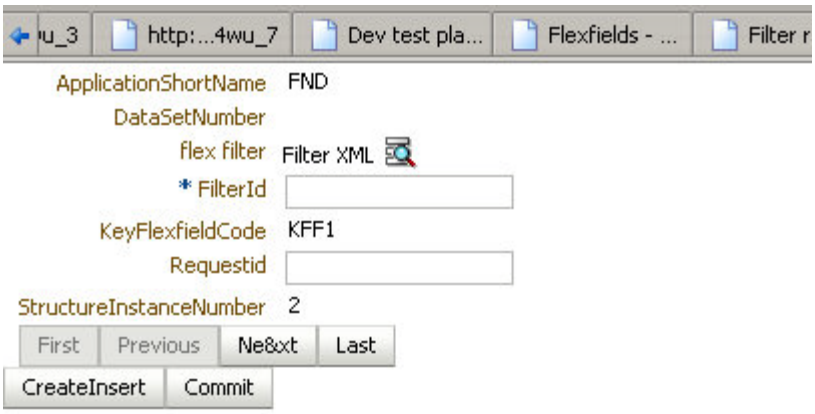
Figure 23–43 Filter Dialog When RestrictConjunctionToAND is Set to true



23.6.3.2 What Happens When You Add a Filter Repository Filter to an Application Page

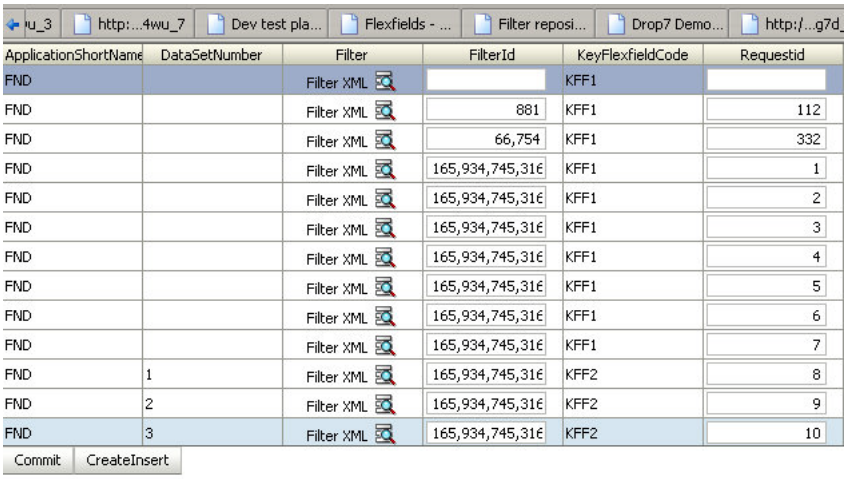
When you add a filter based on the public entity object `FndKfEssFiltersPEO` to your application page as an ADF Form component, the resulting Oracle BI Publisher report submission user interface appears as shown in Figure 23–44.

Figure 23–44 Form-Based Report Submission Combination Filter UI



When you add a filter-repository filter to the application page as an ADF Table component, the report submission user interface appears as shown in Figure 23–45.

Figure 23–45 Table-Based Report Submission Combination Filter UI



When you click **CreateInsert**, a new row is added which 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 23–11.

Table 23–11 Filter Repository Filter Attribute Columns

Column	Description
KeyFlexfieldCode	The code identifying the key flexfield to which this filter will be applicable.This is a read-only value.

Table 23–11 (Cont.) Filter Repository Filter Attribute Columns

Column	Description
StructureInstanceNumber	This is the SIN, the discriminator attribute for the key flexfield which is used in the key flexfield filter. While creating a new filter definition or submitting a new job, a valid value should be defaulted for this attribute at the view object level. The SIN is required for capturing the filter XML. This is a read-only attribute.
DataSetNumber	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 defaulted in the filter view object. This is a read-only attribute.
FilterId	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.
ApplicationShortName	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 **ApplicationShortName** to be `FND`. This is a read-only attribute.
Filter	This is the XML attribute containing the WHERE condition that is set for a particular **FilterId**. The WHERE 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 WHERE condition.

When you click **Commit**, the new row is inserted in the `FND_KF_ESS_FILTERS` database table.

23.6.4 How to Create Combination Filter Definitions for Testing

For testing, you can use INSERT scripts to insert predefined XML filter criteria into the XMLType filter column of your application table.

> **Note:** You can insert this data at any time after the XMLType column has been added to the application 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 key flexfield 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, GREATERTHAN, LESSTHAN, 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_DESCENDENT_OF, IS_LAST_DESCENDENT_OF, IS_PARENT_OF, IS_ANCESTOR_OF, IS_FIRST_ANCESTOR_OF, IS_SIBLING_OF.

For more information about trees, see Chapter 19, "Organizing Hierarchical Data with Tree Structures."

Example 23–27 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'.

Example 23–27 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"
xsi:noNamespaceSchemaLocation="http://www.oracle.com/apps/fnd/applcore/filter/FndF
ilter.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>
              </property>
           </properties>
        </filterCriteriaItem>
        <properties>
           <property>
              <name>TestProp</name>
              <value>ValueProp</value>
           </property>
        </properties>
     </filterCriteriaRow>
  </KeyFlexFilter>
</FndFilter>'));
```

Example of NOTEQUALTO filter:

```
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"
```

```
xsi:noNamespaceSchemaLocation="http://www.oracle.com/apps/fnd/applcore/filter/FndF
ilter.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>
     <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>
  </KeyFlexFilter>
</FndFilter>'));
```

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.

23.6.5 How to Apply Combination Filters Using the PL/SQL Filter APIs

You can take advantage of key flexfield combination filters (including filter-repository filters) without using them in your application user interface. You use the WHERE clause API for standard and cross-validation combination filters, and you use the XML Publisher API for filter repository filters.

23.6.5.1 Applying Standard Filters Using the WHERE Clause API

Applications Core Technology has provided 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 snippet, 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 snippet 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 23–28.

Example 23–28 WHERE Clause Signature

```
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 takes the Filter as XMLType in parameter, tablealias,
--  bindprefix and computes the WHERE clause associated to the filter
--  and provides it as out parameter
--  Params
--     IN Params
--         filter      XMLType  - Filter to be converted to sql snippet
--         tableAlias  Varchar2 - Alias table name to be used in Sql snippet
--         bindPrefix  Varchar2 - Bind Prefix
--      OUT Params
--         sin         Number         - Structure Instance Number
--         bindValues  BIND_VAL_TAB   - List of Bind Values
--         filterWhereClause Varchar2 - Whereclause sql snippet
-----------------------------------------------------------------------------*/
```

The bind values are defined as shown in Example 23–29.

Example 23–29 Bind Values Definition

```
create or replace PACKAGE FND_KF_COMBINATION_FILTER_API AS

  VARCHAR_TYPE CONSTANT Varchar2(20) :='VARCHAR2';
  NUMBER_TYPE  CONSTANT Varchar2(20) :='NUMBER';
  DATE_TYPE    CONSTANT Varchar2(20) := 'DATE';

TYPE BIND_VALUE IS RECORD(
         NAME            Varchar2(30),
         TYPE            Varchar2(20),
         VALUE_VARCHAR2  Varchar2(32767),
         VALUE_NUMBER    Number,
         VALUE_DATE      Date);

 TYPE BIND_VAL_TAB IS TABLE OF BIND_VALUE INDEX BY BINARY_INTEGER;
```

Example 23–30, Example 23–31, and Example 23–32 demonstrate how to use the WHERE clause API for an EQUALTO condition, a BETWEEN condition, and multiple conditions.

Example 23–30 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:

```
FND_KF_COMBINATION_FILTER_API.BuildWhereClause(
                              filter=>v_filter,
                              tableAlias => 'FKFF1',
                              bindPrefix => 'BND',
                              sin => v_sin,
                              bindValues => v_bind,
                              filterWhereClause => v_query);
```

The `tableAlias` value should be used in `WHERE` clause snippets to represent the combinations table name, so in this example,

`FND_KF_TEST_CCT1``.SEGMENT1_VARCHAR2`

should be entered as

`FKFF1``.SEGMENT1_VARCHAR2`

Similarly, the `bindPrefix` value should be used 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:

```
select code_combination_id,
       Segment1_VARCHAR2
from FND_KF_TEST_CCT1 FKFF1
where FKFF1.structure_instance_number=12
and  FKFF1.SEGMENT1_VARCHAR2=123
```

Example 23–31 Using the WHERE Clause API for a BETWEEN Condition

The following listing shows an example of a BETWEEN operator used as part of a filter expression of the form "*attribute* BETWEEN *value1* AND *value2*".

```
<?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>
      <attributeName>_P6_S0</attributeName>
      <columnName>SEGMENT1_Varchar2</columnName>
      <operator>BETWEEN</operator>
      <conjunction>AND</conjunction>
      <valueDataType>NUMBER</valueDataType>
      <value>-500</value>
      <value>1000</value>
```

```
    </filterCriteriaItem>
  </filterCriteriaRow>
</KeyFlexCodeCombinationFilter>'));
```

The Filter expression captured in the preceding XML resolves to the following:

```
SEGMENT1_VARCHAR2 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.SEGMENT1_VARCHAR2 = :BND1
sin               - 12
bindValues        - bindValues(1).NAME = BND1
                  - bindValues(1).TYPE = VARCHAR2
                  - bindValues(1).VALUE_VARCHAR2 = -500
                  - bindValues(2).NAME = BND2
                  - bindValues(2).TYPE = VARCHAR2
                  - bindValues(2).VALUE_VARCHAR2 = 1000
```

With this output you can assemble the following WHERE clause for a BETWEEN filter condition:

```
select code_combination_id,
       Segment1_VARCHAR2
from FND_KF_TEST_CCT1 FKFF1
where FKFF1.structure_instance_number=12
and  FKFF1.SEGMENT1_VARCHAR2 BETWEEN -500 AND 1000
```

Example 23–32 Using the WHERE Clause API for Multiple Conditions

The following listing shows an example of multiple operators used as part of a filter expression of the form "*attribute1* EQUALTO *value1* AND *attribute2* EQUALTO *value2*."

```
<?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/Ke
yFlexCodeCombinationFilter.xsd">
  <keyFlexfieldCode>KFF1</keyFlexfieldCode>
  <structureInstanceCode>VS_FRM_NUM_ON_CHR</structureInstanceCode>
  <applicationShortName>FND</applicationShortName>
  <filterCriteriaRow>
    <filterCriteriaItem>
      <attributeName>_P6_S0</attributeName>
      <columnName>SEGMENT1_Varchar2</columnName>
      <operator>EQUALTO</operator>
      <conjunction>AND</conjunction>
      <valueDataType>NUMBER</valueDataType>
      <value>123456</value>
    </filterCriteriaItem>
```

```
    <filterCriteriaItem>
      <attributeName>_P6_S2</attributeName>
      <columnName>SEGMENT2_Varchar2</columnName>
      <operator>EQUALTO</operator>
      <conjunction>AND</conjunction>
      <valueDataType>NUMBER</valueDataType>
      <value>123.45</value>
    </filterCriteriaItem>
  </filterCriteriaRow>
</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:

```
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:

```
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
```

23.6.5.2 Applying 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 23–33.

Example 23–33 kff_filter Signature

```
/* 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 23–34 demonstrates how to use this API to obtain the where clause and bind variable information for a filter in the filter repository.

Example 23–34 Using the Filter Repository API

```
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;
```

23.6.6 How to Remove Combination Filters from Your Application

To remove a key flexfield 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, you will be presented with a list of those filter accessors. Select the one that you want to remove.

23.6.7 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 23–35.

Example 23–35 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');
```

24

Using 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:

24.1 Introduction to Extensible Flexfields

An extensible flexfield is similar to a descriptive flexfield in that it provides a customizable expansion space that implementers, 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 a child table. For a general introduction to flexfields, segments, and contexts see Chapter 21, "Getting Started with Flexfields." For more details about the differences between descriptive flexfields and extensible flexfields, see Section 24.1.2, "The Benefits of Extensible Flexfields."

Implementers can combine and arrange the segments into *contexts* (attribute groups) that are tailored to a company's specific needs. For example, they can group related segments so that they appear together on the page. In addition, you can optionally set up an extensible flexfield to enable implementers to group contexts into categories. To understand how implementers configure extensible flexfields to meet a company's

specific needs, see the "Using Flexfields for Custom Attributes" in the *Oracle Fusion Applications Extensibility Guide*.

24.1.1 Understanding Extensible Flexfields

Extensible flexfields are comprised of the following key artifacts:

- Contexts (attribute groups)
- Context-sensitive segments
- Logical pages
- Categories
- Category hierarchies
- Usages

24.1.1.1 About Contexts (Attribute Groups)

Extensible flexfield contexts are a data grouping mechanism that implementers can use to arrange segments into meaningful groups. Each context is a group of attributes that displays in its own subregion of the user interface page at runtime. Implementers create and configure the contexts. After creating a context, an implementer 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 learn about categories in Section 24.1.1.4, "About Categories."

Figure 24–1 shows the user interface 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 24–2 and Figure 24–3 show user interface 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 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 custom configures the extensible flexfield that is embedded in it.

Extensible flexfields allow implementers 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 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. Certification and

License Requirements is a multi-row context, so end users can enter multiple rows, one for each certificate or license required for the position.

Figure 24–1 Position User Interface Page

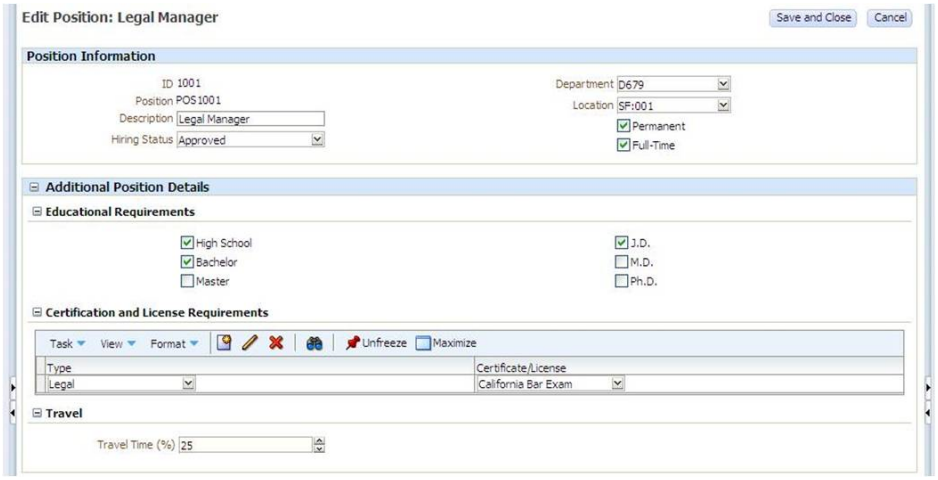


Figure 24–2 Parts User Interface Page for the All-in-One Printers Category

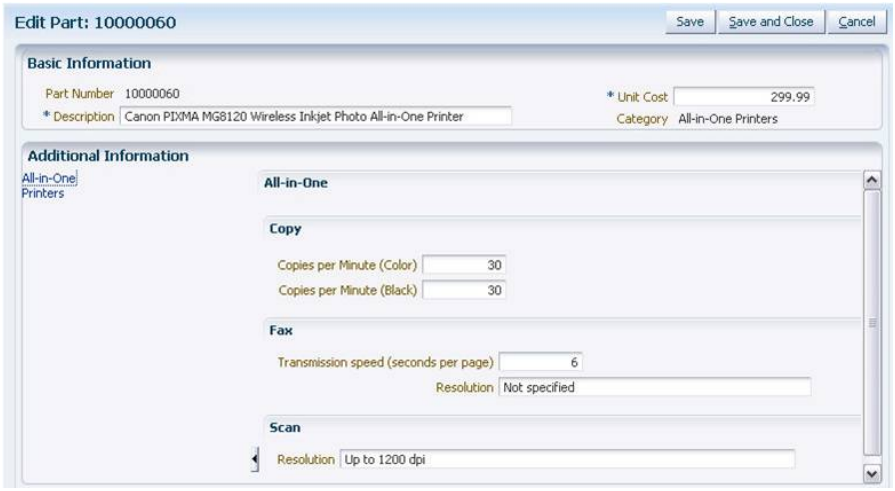
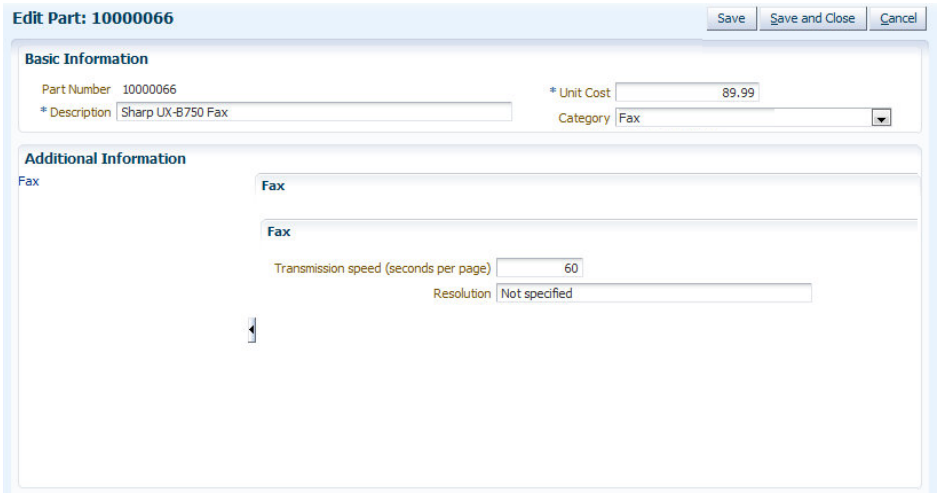


Figure 24–3 Parts User Interface Page for the Fax Category



24.1.1.2 Context-Sensitive Segments

With extensible flexfields, every segment is a member of a context (attribute group). That is, all segments are context-sensitive. Implementers define the contexts and their context-sensitive segments. Context-sensitive segments are the lowest-level data points that implementers 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 render as ADF Faces rich client components, such as text box, text area, LOV choice list, date picker, check box, 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.

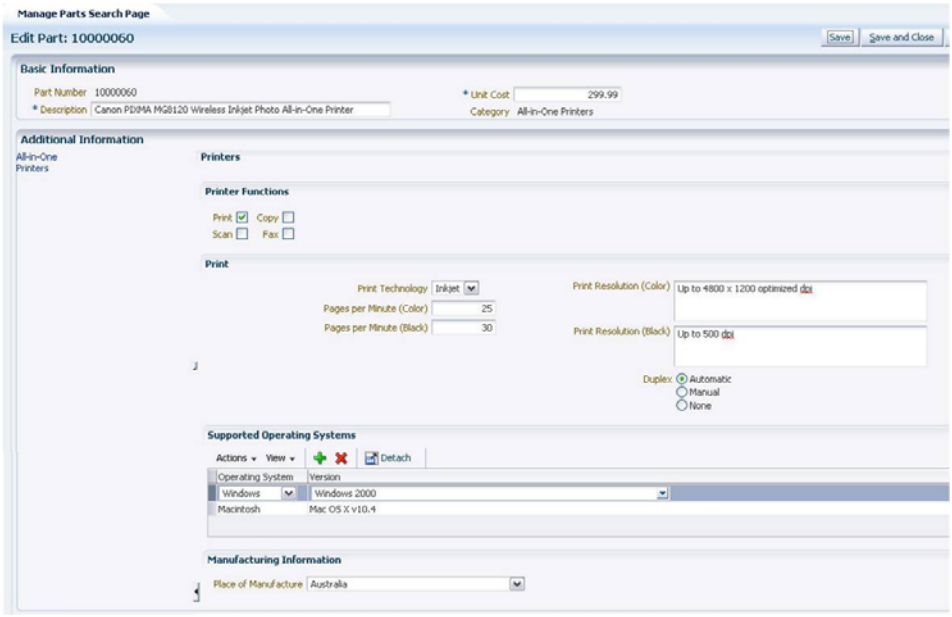
24.1.1.3 About Logical Pages

Extensible flexfields enable implementers 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 implementers associate the logical pages with categories on a flexfield usage basis. You learn about usages in Section 24.1.1.6, "About Usages (Data Levels)."

In the Parts user interface page shown in Figure 24–2, an implementer has defined two logical pages for this category — Printers and All-in-One. The implementer 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 pane. End users view a logical page in the right pane by selecting it from the list. In this figure, the end user has chosen to view the All-in-One logical page. You can see the Printers logical page in Figure 24–4.

Figure 24–4 Printers Logical Page in the Parts User Interface Page



Note that you have different layout options for the logical pages. For example, in Figure 24–1, the developers chose to hide the entire left 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 pane. For more information about layout options, see Section 24.6, "Employing Extensible Flexfields on an Application 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 differentiator, 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 implementers 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.

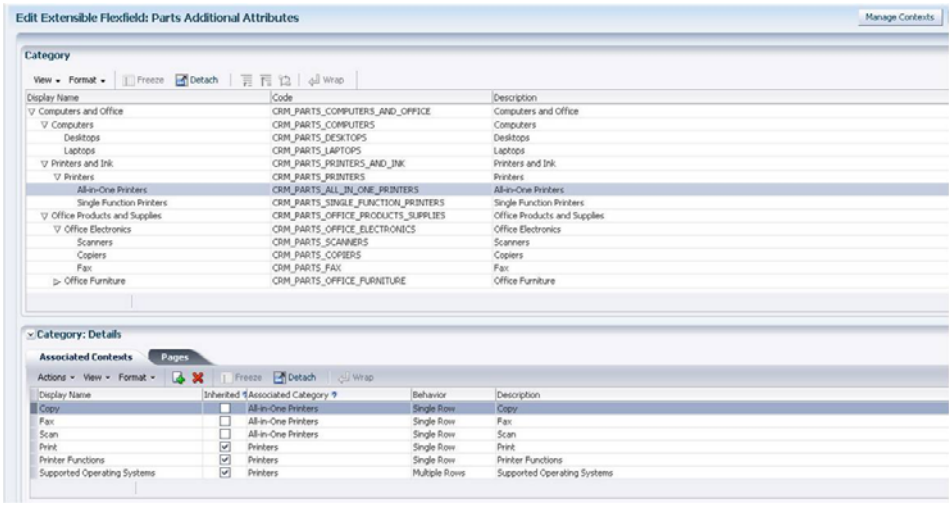
In Figure 24–2 and Figure 24–3 the segments that the region displays depend on which category the part belongs to. In Figure 24–2, the part belongs to the All-in-One Printers category, and, in Figure 24–3, the part belongs to the Fax category.

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 implementers. Figure 24–5 shows an example of categories defined in a hierarchical fashion. In this example, the root category is Computers and Office. One of its child categories is Printers and Ink. The Printers and Ink category, in turn, has a child category Printers. The Printers

category has two children categories — All-in-One Printers and Single Function Printers.

Figure 24–5 Example of a Category Hierarchy



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, Fax, 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. Implementers 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 need to 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 implementers 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 implementers by using extensible flexfields:

- Custom grouping: Extensible flexfields enable implementers to group segments into contexts (attribute groups), and each context is displayed in its own region.
- Multiple logical pages: When an implementer defines a large number of custom attributes, the implementer 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 implementers 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 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 their parent categories.
- Reusable attribute groups: When you enable multiple categories for an extensible flexfield, implementers can associate contexts with more than one category. When their business object instances share common attributes, the implementers can put the common attributes in one context and reuse that context for in all applicable categories, thus minimizing set-up 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 implementer can define for a context is limited to the number of underlying flexfield database columns, there is no limit to the number of contexts that an implementer can create for an extensible flexfield. Implementers can use the flexfield database columns over and over in newly created contexts resulting in an unlimited number of custom attributes.
- Custom lists: When implementers 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 implementers to store different attribute values on a locale-by-locale basis. For example, an implementer 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.

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 artifacts and user interface task flows.

24.1.3 Extensible Flexfield Structure and Content

Unlike descriptive flexfields which are mapped to extension columns in the base application table, extensible flexfields are mapped to extension columns in extension tables that are separate from the base application 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. The category hierarchy translates into a hierarchy of polymorphic view objects with view links to associated context view objects. A CategoryCode attribute acts as the discriminator that determines which category view row type should be 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 based upon the category. An application module, which holds the category view object, is generated for each category.

For more information about polymorphic view rows, see the "Working with Polymorphic View Rows" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

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 flexfield 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 base application table for which you are implementing the extensible flexfield. For example, you might want to enable implementers to create custom attributes for the Parts business object, which corresponds to the FND_CRM_PARTS table.
2. Decide whether 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 whether you want to enable customers to store the information for a context at different levels (also referred to as flexfield usages). For example, implementers 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 user interface and the part interface, 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. Ascertain whether 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 flexfields 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 table and a view of the translation table for each usage as well.

   See Section 24.3, "Creating Extensible Flexfield Data Tables."

2. Create the flexfield metadata. In this step, you use PL/SQL procedures to register the flexfield and its usages, create a default context data element, and register the flexfield data columns and root category.

   See Section 24.4, "Defining and Registering Extensible Flexfields."

3. Create and configure business components to support the extensible flexfield. In this step, you create an entity object and a view object over the base extension table, and, if they exist, the translation extension table and view. You then create the base category view object over the base application table entity object (the table for which the flexfield is an extension). If you want to enable search capabilities, you create a declarative view object for searching over the base application table that the flexfield extends. Last, you configure the flexfield usages' application modules to support flexfields and you register the flexfield's business components.

   See Section 24.5, "Defining and Registering Extensible Flexfield Business Components."

4. Add the extensible flexfield to the appropriate application pages.

   See Section 24.6, "Employing Extensible Flexfields on an Application Page."

5. Load any necessary application seed data, such as error messages and lookup values.

   See Section 24.7, "Loading Seed Data."

6. Optionally, customize the generated model or the generated user interface artifacts (or both).

   See Section 24.8, "Customizing the Extensible Flexfield Runtime Business Component Modeler" and Section 24.9, "Customizing the Extensible Flexfield Runtime User Interface Modeler."

7. Test the flexfield.

   See Section 24.10, "Testing the Flexfield."

Once you have completed the extensible flexfield development process and delivered your application, implementers 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, such as defining attributes, labels, behavior, and associated value sets, see the "Using Flexfields for Custom Attributes" chapter in the *Oracle Fusion Applications Extensibility Guide*.

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 implementers, you register it with *Oracle Fusion Functional Setup Manager*. For more information, see Section 25.5, "Integrating Flexfield Task Flows into Oracle Fusion Functional Setup Manager".

24.3 Creating 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. An extensible flexfield must have at least one set of tables, which defines the master usage. The implementers will expose only the flexfield attribute columns that they require.

Note: The steps in this section assume that the base application table that the flexfield is extending already exists.

24.3.1 How to Create Extensible Flexfield Database Tables

Each flexfield usage requires a base extension table. This table stores non-translatable contexts and their segment values. Table 24–1 lists the fields that you must include in a base extension table. You can add as many attribute columns — ATTRIBUTE_CHAR*n*, ATTRIBUTE_NUMBER*n*, ATTRIBUTE_NUMBER*n*_UOM, and ATTRIBUTE_TIMESTAMP*n*, as you feel is necessary. The numbers of attribute columns shown in the table should be adequate for most cases. The table name should have a suffix of _B to identify it as the base 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 view over the translation extension table. The translation extension table stores the translatable contexts and their segment values. The translation extension view returns the rows in the current user locale. Table 24–2 lists the required translation extension table columns and Table 24–3 lists the required translation extension view columns. The translation table name should have a suffix of _TL to identify it as a translation table, and the name of the view of the translation table should have a suffix of _VL.

Create one `VARCHAR2` type attribute column of the same name in the translation extension table for each corresponding ATTRIBUTE_CHAR*n* column in the base extension table, such as ATTRIBUTE_CHAR1. The translation extension view should contain the same set of ATTRIBUTE_CHAR*n* columns as the translation extension table.

At runtime, as end users enter attribute values for each context, the context and its attribute values are stored as a row in the base extension table if the context is not translatable, or as a row in the translation extension table if the context is translatable. When the flexfield is deployed, each context is translated into a business component view object over the columns in the extension table.

You must use the Database Schema Deployment Framework tools to create the application table and columns. Using these tools ensures that the table and its columns are registered in the Applications Core Data Dictionary. For more information, see Chapter 56, "Using the Database Schema Deployment Framework."

Note: To avoid any compatibility and interoperability problems, you should format the column names as indicated.

Table 24–1 Extensible Flexfield Base Extension Table (_B) Specification

Column	Type	Nullable?
EFF_LINE_ID	NUMBER(18)	No
OBJECT_VERSION_NUMBER	NUMBER(9)	No
Primary key columns of the application table for which this extensible flexfield is being defined.	Same as the column in the application.	No
CONTEXT_CODE	VARCHAR2(80)	No
CREATED_BY	VARCHAR2(64)	No
CREATION_DATE	TIMESTAMP(6)	No
LAST_UPDATED_BY	VARCHAR2(64)	No
LAST_UPDATE_DATE	TIMESTAMP(6)	No
LAST_UPDATE_LOGIN	VARCHAR2(32)	Yes
ATTRIBUTE_CHAR1	VARCHAR2(150)	Yes
...		
ATTRIBUTE_CHAR40	VARCHAR2(150)	Yes
ATTRIBUTE_NUMBER1	NUMBER	Yes
ATTRIBUTE_NUMBER1_UOM	VARCHAR2(9)	Yes
....		
ATTRIBUTE_NUMBER20	NUMBER	Yes
ATTRIBUTE_NUMBER20_UOM	VARCHAR2(9)	Yes
ATTRIBUTE_TIMESTAMP1	TIMESTAMP	Yes
...	TIMESTAMP	Yes
ATTRIBUTE_TIMESTAMP10	TIMESTAMP	Yes
PROGRAM_NAME	VARCHAR2(30)	Yes
PROGRAM_APP_NAME	VARCHAR2(50)	Yes

Table 24–2 Extensible Flexfield Translation Extension Table (_TL) Specification

Column	Type	Nullable?
EFF_LINE_ID	NUMBER(18)	No
OBJECT_VERSION_NUMBER	NUMBER(9)	No
Primary key columns of the application table for which this extensible flexfield is being defined.	Same as the column in the application.	No
CONTEXT_CODE	VARCHAR2(80)	No
CREATED_BY	VARCHAR2(64)	No
CREATION_DATE	TIMESTAMP(6)	No
LAST_UPDATED_BY	VARCHAR2(64)	No
LAST_UPDATE_DATE	TIMESTAMP(6)	No
LAST_UPDATE_LOGIN	VARCHAR2(32)	Yes

Table 24–2 (Cont.) Extensible Flexfield Translation Extension Table (_TL) Specification

Column	Type	Nullable?
SOURCE_LANG	VARCHAR2(4)	No
LANGUAGE	VARCHAR2(4)	No
ATTRIBUTE_CHAR1	VARCHAR2(1000)	Yes
...	VARCHAR2(1000)	Yes
ATTRIBUTE_CHAR40	VARCHAR2(1000)	Yes
PROGRAM_NAME	VARCHAR2(30)	Yes
PROGRAM_APP_NAME	VARCHAR2(50)	Yes

Table 24–3 Extensible Flexfield Translation Extension View (_VL) Specification

Column	Type	Nullable?
EFF_LINE_ID	NUMBER(18)	No
Primary key columns of the application table for which this extensible flexfield is being defined.	Same as the column in the application.	No
CONTEXT_CODE	VARCHAR2(80)	No
ATTRIBUTE_CHAR1	VARCHAR2(1000)	Yes
...		
ATTRIBUTE_CHAR40	VARCHAR2(1000)	Yes
PROGRAM_NAME	VARCHAR2(30)	Yes
PROGRAM_APP_NAME	VARCHAR2(50)	Yes

24.4 Defining and Registering Extensible Flexfields

After you create a set of dedicated tables for a flexfield's usage, you use procedures from the FND_FLEX_DF_SETUP_APIS PL/SQL package to register the flexfield usage and define its metadata in seed tables.

The definition of an extensible flexfield usage includes the following information:

- The logical name of the flexfield.
- The name of the database table containing the columns to be used as flexfield segments.
- The names of the database table columns to be used as flexfield segments.
- The identity of the flexfield context segment that will indicate how context-sensitive segments are used in each data row.
- The name of the custom modeler in fnd_df_flexfields_b.ADFBC_MODELER.
- Flexfield usage information, including Oracle Metadata Service (MDS) root packages for usages — the name of the root package for all business components that model the flexfield. Each usage can have its own package name.

After the implementers 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 belonging to the extensible flexfield, and the contexts associated with each category, page and usage.
- Validation rules associated with the segments. Each segment can have its own LOV or validation rules.

24.4.1 How to Register Extensible Flexfields

You use the `FND_FLEX_DF_SETUP_APIS` PL/SQL API package to register new extensible flexfield usages and to add the specified flexfield metadata to the following global configuration metadata.

- FND_DF_FLEXFIELDS_B, FND_DF_FLEXFIELDS_TL

  Add the extensible flexfield's code name to both tables.

- FND_DF_FLEX_USAGES_B, FND_DF_FLEX_USAGES_TL

  Add the extensible flexfield usages.

- FND_DF_FLEX_USAGES_TL

  Add the display names for the flexfield usages.

- FND_DF_TABLE_USAGES

  For each usage, add the base application table name for which the flexfield is implemented and set the table type to BASE. Also add the usage base table name with a type of EXTENSION, and add the usage translation table name with a type of EXTENSION_TL. Set the table usage code to the flexfield's usage code.

  If the consumers want to support interface loading of extensible flexfield data, also add entries for the BASE_INTERFACE, EXTENSION_INTERFACE, and EXTENSION_INTERFACE_TL table types.

- FND_DF_SEGMENTS_B

  Add one row for the extensible flexfield with CONTEXT_CODE set to "Context Data Element" and SEGMENT_CODE set to "Context Segment."

- FND_FF_COLUMN_USAGES

  For each usage, add an entry for each context-specific column in the EXTENSION and EXTENSION_TL tables. For example, add entries for all the ATTRIBUTE_CHAR*n* columns, for all the ATTRIBUTE_NUMBER*n* columns, and so forth.

- FND_EF_CATEGORIES_B

  An extensible flexfield requires at least one entry in this table. Add additional entries only if you are shipping your application with some categories already defined.

After you create the flexfield's business components, you must complete the registration process by adding entity details, as described in Section 24.4.1, "How to Register Extensible Flexfields." You must have at least one entity usage per base extension table. If you require parallel sets of extensible flexfield artifacts generated for different uses (for example, public view objects and private view objects), you can replicate the entity usage entries with a different group name. The flexfield business component modeler generates a separate set of components for each group.

For more information, refer to the package specification.

24.5 Defining and Registering Extensible Flexfield Business Components

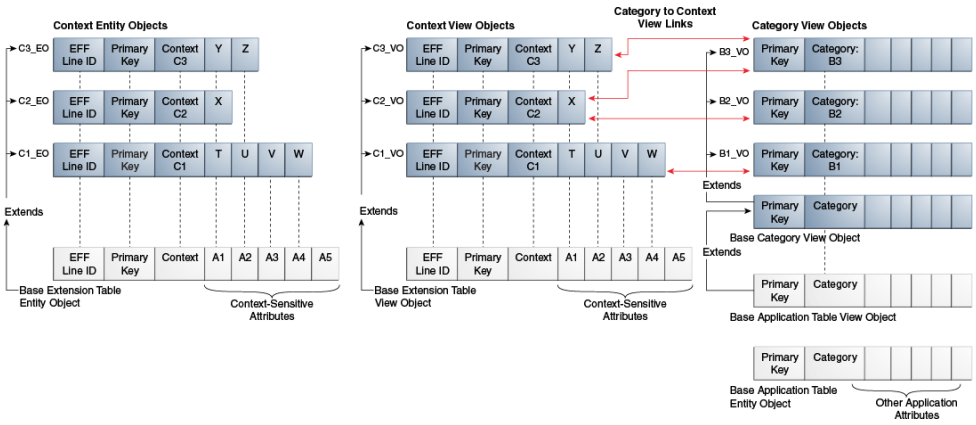
To incorporate an extensible flexfield into your application, you must define and configure entity objects and view objects for each set of database tables that are defined for each flexfield usage.

Figure 24–6 shows the extensible flexfield business components for a base application table. In this example, the flexfield is not translatable. The developer creates an entity object and view object over the base extension table to support contexts.

If the flexfield is translatable, the developer would also create an entity object and view object over the translation extension table and the translation extension view.

When an implementer deploys flexfield configurations, the deployment process creates the extended context entity objects, context view objects, category view objects, and category-to-context view links. Notice that in this example, the C2 context is a member of two categories.

Figure 24–6 Extensible Flexfield Business Components



To define an extensible flexfield business component:

1. Create and configure extensible flexfield entity objects over the base extension table. If a translation extension table exists for the flexfield, create and configure extensible flexfield entity objects over the translation extension table and the translation table view.
2. Configure the `EFF_LINE_ID` attribute for each extensible flexfield entity object as a unique ID.
3. Create and configure flexfield view objects to support contexts, categories, and, optionally, searching.
4. For each extensible flexfield usage, configure its application module to support extensible field usage by adding a `createDetailRowIfNotExist` method.
5. Register the flexfield business components.

> **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 Section 25.4, "Regenerating Flexfield Business Components Programmatically."

24.5.1 How to Create and Configure Extensible Flexfield Entity Objects

For each extensible flexfield usage, you must create entity objects over the base extension table. If you created a translation extension table and a translation extension view for a usage, you must create entity objects for them as well.

For more information about creating entity objects, see the "Creating a Business Domain Layer Using Entity Objects" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

> **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 `adf-config.xml` and the flexfield usage metadata).

Before you begin:

1. Create one set of dedicated flexfield tables per usage, as described in Section 24.3.1, "How to Create Extensible Flexfield Database Tables."
2. Register the flexfield's usages as described in Section 24.4.1, "How to Register Extensible Flexfields."
3. Ensure that the Applications Core library has been added to the data model project.
4. Verify that at least one customization class is included in `adf-config.xml`. 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 *Oracle Fusion Applications Extensibility Guide*.

24.5.1.1 Creating and Configuring an Entity Object Over the Base Extension Table

For each extensible flexfield usage, use the standard wizard to create an entity object over 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.

To create and configure an entity object over 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 application table
   - `CONTEXT_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 unique 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 *`objectname`*`BEOImpl` entity object class to be generated extends `oracle.apps.fnd.applcore.oaext.model.OAEntityImpl`.
4. After creating the entity object, configure all of its attributes to be hidden.

   On the UI Hints tab of the Property Inspector for each attribute, set the **Display Hint** property to `Hide`.

24.5.1.2 Creating and Configuring an Entity Object Over the Translation Extension Table

If you created a translation extension table for the flexfield usage, as described in Table 24–2, use the standard wizard to create an entity object over the translation extension table, but apply the changes described in the following procedure to support the extensible flexfield.

To create and configure an entity object over 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 application table
   - `CONTEXT_CODE`
   - `SOURCE_LANG`
   - `LANGUAGE`
   - `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 unique 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 *`objectname`*`TlEOImpl` entity object class to be generated extends `oracle.apps.fnd.applcore.oaext.model.OAEntityImpl`.
4. After creating the entity object, configure all of its attributes to be hidden.

   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`.

24.5.1.3 Creating and Configuring an Entity Object Over 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 over the translation extension view that is described in Table 24–3, but apply the changes described in the following procedure to support the extensible flexfield.

To create and configure an entity object over 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 application table
   - `CONTEXT_CODE`
2. On the Attribute Settings page, do the following:
   - Set the `EFF_LINE_ID` attribute to be a unique 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 *`objectname`*`VlEOImpl` entity object class to be generated extends `oracle.apps.fnd.applcore.oaext.model.OAEntityImpl`.
4. After creating the entity object, configure all of its attributes to be hidden.

   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.

24.5.2 How to Configure the EFF_LINE_ID Attribute as a Unique ID

After you create and configure entity objects over the base extension table, translation extension table, and view of the translation extension table as described in Section 24.5.1, "How to Create and Configure Extensible Flexfield Entity Objects," you must configure the `EFF_LINE_ID` attribute for each entity object as a unique ID.

The Unique ID Generator provides a way for your application to generate unique IDs for entity object attributes of type `BigDecimal`. The unique IDs generated are also of type `BigDecimal`, and meet certain required criteria for uniqueness across database instances.

To configure EFF_LINE_ID as a unique ID:

1. Create a connection to the database that contains the Unique ID Generator table called S_ROW_ID.
2. Configure a special default-value expression for `EFF_LINE_ID` that will invoke the Unique ID Generator when needed.

For more information, see Section 9.6, "Using Unique ID."

24.5.3 How to Create and Configure Extensible Flexfield View Objects

For each extensible flexfield usage, use the standard 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 over the base extension table entity object and, if the flexfield has a translation extension table, one view object over the translation extension view entity object.

  Note: No view objects are required over the translation table entity object.

- To support categories for the flexfield, create a view object over the base application table entity object for which you are developing this extensible flexfield.
- To support searching, create declarative view objects.

For more information about creating view objects, see the "Defining SQL Queries Using View Objects" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Before you begin:

1. Create and configure entity objects over the base extension table, translation extension table, and view as described in Section 24.5.1, "How to Create and Configure Extensible Flexfield Entity Objects."
2. Configure the `EFF_LINE_ID` attribute as a unique ID as described in Section 24.5.2, "How to Configure the EFF_LINE_ID Attribute as a Unique ID."

24.5.3.1 Creating and Configuring Context View Objects

To support contexts for an extensible flexfield usage, the usage must have a view object over its base extension table entity object. If you created a translation extension table for the usage, you must also create a view object over the translation extension view entity object.

To create and configure the context view objects:

1. Create the view objects over the base extension table entity object and, if you created a translation extension table for the usage, over the translation extension view entity object in the normal manner.
2. On the Java page for each view object, ensure that the view object classes to be generated extend base classes as follows:
   - *vo_name*`Impl` extends `EFFViewObjectImpl`
   - *vo_name*`RowImpl` extends `EFFViewRowImpl`
   - *vo_name*`DefImpl` extends `EFFViewDefImpl`

   These base classes are in the `oracle.apps.fnd.applcore.oaext.model` package.

24.5.3.2 Creating and Configuring Category View Objects

To support categories for the flexfield, there must be a view object over the base application table entity object for which you are developing the extensible flexfield.

To create and configure the category view object:

1. Create the view object over the base application table for which you are implementing the extensible flexfield in the normal manner. For example, in Figure 24–2, the base application 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 can be entity-based, SQL-derived, or a transient attribute. This attribute should be a VARCHAR2 with a maximum of 800 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.

2. Implement the following interface on the Java tab of the view object definition:

   `oracle.apps.fnd.applcore.oaext.model.EFFCategoryViewObjectInterface`

   Create a method called `queryCategoryRowByPrimaryKey` that will take in up to five primary key values and return the correct row of the product view object as demonstrated in Example 24–1.

   For this example, the view object has a view criteria defined called `getItemByKey`, and bind parameters defined for two primary key attributes, called `OrganizationIdBind` and `InventoryItemIdBind`.

Example 24–1 Query Category Row Method

```
public Row queryCategoryRowByPrimaryKey(
  String pk1, String pk2, String pk3,
  String pk4, String pk5)
    {
        Long invId = new Long (pk1);
        Long orgId = new Long (pk2);

        setWhereClauseParams(null);
        applyViewCriteria(null);

        ViewCriteria vc = getViewCriteria("getItemByKey");
        setNamedWhereClauseParam("OrganizationIdBind", orgId);
        setNamedWhereClauseParam("InventoryItemIdBind", invId);

        applyViewCriteria(vc);
        executeQuery();
        Row r = null;
        if(hasNext())
        {
          r = next();
          setCurrentRow(r);
        }
        return r;
    }
```

This method queries the correct category row based on up to five primary key values passed into the method.

24.5.3.3 Creating Declarative View Objects for Searching

To enable searching, there must be a declarative view object for searching over the base application table for which you are implementing the extensible flexfield.

To create declarative view objects for searching:

1. Create the declarative view object in the normal manner.
2. On the Java page, confirm that the *objectname*`VOImpl` view object class to be generated extends the following class:

   `oracle.apps.fnd.applcore.oaext.model.EFFCategoryViewObjectImpl`

   The `EFFCategoryViewObjectImpl` class adds the additional `WHERE` clause that provides security.

24.5.4 How to Configure an Extensible Flexfield Application Module

To enable application pages that contain a flexfield usage to display the flexfield's segments, you must configure an application module to support that flexfield usage.

You can use the application module that you have already created for your application, or you can use the standard wizard to create a new application module dedicated to your extensible flexfield.

To the application module's Java client interface, add a method called `createDetailRowIfNotExist`, as shown in Example 24–2.

Example 24–2 Java Source for createDetailRowIfNotExist Method

```
public void createDetailRowIfNotExist(
  String categoryAMInstName, String categoryViewUsageName,
  String contextViewLinkAccName, String category_pk1,
  String category_pk2, String category_pk3,
  String category_pk4, String category_pk5)
{
  EffCategoryAMImpl effAM = (EffCategoryAMImpl)
    findApplicationModule(categoryAMInstName);
  effAM.createDetailRowIfNotExist(
    categoryViewUsageName, contextViewLinkAccName,
    category_pk1, category_pk2, category_pk3,
    category_pk4, category_pk5);
}
```

At implementation, the extensible flexfield runtime business component modeler creates a personalization on this application module, and generates an application module instance for each category that the implementer defines.

24.5.5 How to Register the Extensible Flexfield Business Components

After you configure business components to support the extensible flexfield, you must complete the flexfield registration process by providing the names of the flexfield's entity object and view object. If you created a translation table for the flexfield, you must also register the entity object over the translation table.

You use procedures from the `FND_FLEX_DF_SETUP_APIS` PL/SQL package to register the flexfield entity details. For information about using the procedures, refer to the package specification.

To register extensible flexfield entity details:

Add the flexfield metadata to the FND_DF_ADFBC_USAGES table as specified by the package specification.

For each table defined in FND_DF_TABLE_USAGES, specify the name of the entity object and view object. For the translation table, also specify the name of the entity object over the translation table.

If the consumers want to generate a different set of artifacts for both private and public groups, create an additional set of rows with a different extensible flexfield group name.

Figure 24–7 shows example entries for a flexfield usage. As described in Section 24.4.1, "How to Register Extensible Flexfields," the FND_DF_TABLE_USAGES table contains rows for the base application table (USGA_BASE), the usage base extension table (USGA_EFF_B), and the usage translation 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 usage translation table also name the entity object over the translation table. Note that not all columns are shown.

Note: In this example, the Public group is necessary because the consuming product requires a parallel set of extensible flexfield artifacts generated for public view objects.

If the consumers wanted to support interface loading of extensible flexfield data, you would also add entries for the BASE_INTERFACE, EXTENSION_INTERFACE, and EXTENSION_INTERFACE_TL table types.

Figure 24–7 Example Entity Detail Entries

FND_DF_TABLE_USAGES

DESCRIPTIVE_FLEXFIELD_CODE	FLEXFIELD_USAGE_CODE	TABLE_NAME	TABLE_TYPE	TABLE_USAGE_CODE
ABC_EFF	ABC_USG_A	USGA_BASE	BASE	ABC_USG_A
ABC_EFF	ABC_USG_A	USGA_EFF_B	EXTENSION	ABC_USG_A
ABC_EFF	ABC_USG_A	USGA_EFF_TL	EXTENSION_TL	ABC_USG_A

FND_DF_ADFBC_USAGES

EFF_GROUP_NAME	TABLE_USAGE_CODE	TABLE_NAME	ENTITY_OBJECT	ENTITY_OBJECT_TL	VIEW_OBJECT
Private	ABC_USG_A	USGA_BASE	BaseEO		BaseVO
Private	ABC_USG_A	USGA_EFF_B	EFFBEO		EFFBVO
Private	ABC_USG_A	USGA_EFF_TL	EFFVLEO	EFFTLEO	EFFVLVO
Public	ABC_USG_A	USGA_BASE	BasePEO		BasePVO
Public	ABC_USG_A	USGA_EFF_B	EFFBPEO		EFFBPVO
Public	ABC_USG_A	USGA_EFF_TL	EFFVLPEO	EFFTLPEO	EFFVLPVO

24.6 Employing Extensible Flexfields on an Application Page

You can incorporate an extensible flexfield into an application with several user interface variations:

- As 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 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 a single task flow that presents a set of contexts associated with a specified extensible flexfield logical page, which is identified before the application page initializes. Figure 24–1 is an example of this variation.
- As a single task flow that presents a context, which will be identified before the application page initializes.

Note: When it is not clear what type of data will be seeded for the extensible flexfield, name the containing region, such as a page or a dialog, "Additional Information" or "Additional Information: *Object Name*" for view-only data, and "Edit Additional Information" or "Edit Additional Information: *Object Name"* for editable data. If the containing region is a tab, name the tab "Additional Information" or "Edit Additional Information," as appropriate. This convention ensures consistency across Oracle Fusion applications.

24.6.1 How to Expose the Pages and Contexts Associated with One Extensible Flexfield Usage

You can incorporate an extensible flexfield into an application as a list of the extensible flexfield pages for a single usage, combined with the contexts for the selected page, and integrated into a single task flow.

To expose the pages and contexts that are associated with one extensible flexfield usage:

1. Create a task flow for the single extensible flexfield usage, which includes an application page fragment with a splitter that contains the list of the usage's extensible flexfield pages on the left, and the contexts associated with the selected flexfield page on the right.
2. Add the task flow to the page.
3. Render the page.

24.6.1.1 Creating a Task Flow for a Single Extensible Flexfield Usage

Create a task flow to be added to the page that will display the page lists and contexts.

Before you begin:

Create and configure business components to support the extensible flexfield as described in Section 24.5, "Defining and Registering Extensible Flexfield Business Components."

To create the task flow:

1. Create a task flow with a default view activity pointing to a fragment.
2. Add an `EffCategoryPagesBean` managed bean with the class `oracle.apps.fnd.applcore.flex.eff.runtime.EffCategoryPagesBean` to the task flow. The scope should be `pageFlow`.
3. Open the fragment and drag and drop a splitter control.
4. On the first facet, drag and drop the `EffCatPageListContainer` seeded task flow from the ViewController FlexModeler-View library.
5. Add 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.EffCategoryPagesBean}`
6. Do the same for the Context Container Page 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.EffCategoryPagesBean}`
   - `_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

24.6.1.2 Adding the Task Flow to the Page

Add the task flow for the extensible flexfield usage to the page that will display the page lists and contexts.

Before you begin:

Create the task flow for the extensible usage as described in Section 24.6.1.1, "Creating a Task Flow for a Single Extensible Flexfield Usage."

To add the task flow to the page:

1. Open the JSPX page that will display the page lists and contexts.
2. Drop the task flow on the page.

24.6.1.3 Rendering the Page

Render the page to view the user interface.

Before you begin:

- Add the task flow for the extensible usage to the page as described in Section 24.6.1.2, "Adding the Task Flow to the Page."
- Obtain the flexfield usage's package name from the FND_DF_ADFBC_USAGES.PACKAGE_NAME column.

To render the page:

1. Open the `Databindings.cpx` file for the project that is consuming the generated extensible flexfield task flows and add the `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"/>
```

2. Complete the following steps to mark the flexfield usage's package directory as a flexfield model package:
   a. Create an `inf` directory in the flexfield usage's package directory. For example, if the usage's package is `oracle.apps.fnd.applcore.crmdemo.flex`, then create the `/oracle/apps/fnd/applcore/crmdemo/flex/inf` directory.
   b. To identify the package directory as a flexfield model package, create a file named `FlexfieldPkgInf.xml` in the package's `inf` directory and add the following contents to the file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<flexfield-inf/>
```

Note: Ensure that this directory only contains the automatically generated flexfield MDS files and the `inf` directory with the `FlexfieldPkgInf.xml` file. When you deploy the application, the ANT script that is run by the deployment process will add the required name spaces for the identified package directory to the `adf-config` file, as shown in the following example.

```
<namespace path="/persdef" metadata-store-usage="mdsRepos"/>
<namespace
path="/oracle/apps/scm/productCatalogManagement/items/protectedMode
l/itemRevisionEff"metadata-store-usage="mdsRepos"/>
<namespace
path="oracle/apps/scm/productCatalogManagement/"items/protectedMode
l/itemsEffmetadata-store-usage="mdsRepos"/>
```

3. Complete the following steps to identify the flexfield user interface packages:

   a. Either create a WAR deployment profile in the project that is named `FlexfieldViewController` and that contains flexfield user interface packages, or reference the flexfield user interface packages through a library JAR file. Do not include this deployment profile in the application EAR file. This is just a marker profile.

   b. Find the flexfield user interface package directory under the `ViewController/public_html` directory and add an `inf` subdirectory to the flexfield user interface package directory.

   c. To identify the package directory as a flexfield user interface package, create a file named `FlexfieldViewPkgInf.xml` in the package's `inf` directory and add the following contents to the file.

   ```
   <?xml version="1.0" encoding="UTF-8" ?>
   <flexfield-inf/>
   ```

   Note: The package directory will be added to the `adf-config` file and thus only should contain the automatically generated flexfield MDS files and the `inf` directory with this file.

   Do not repeat these steps for the corresponding `pageDefs` packages under Application Sources. Those packages are added automatically when the flexfield is deployed.

4. Add the following snippet to the `Application` tag at the top of `Databindings.cpx` in order for the application to find the page definitions for the generated user interface artifacts:

   ```
   PageMapClass="oracle.jbo.uicli.mom.DynamicPageMapImpl"
                BasePageDefPackageName="pageDefs"
   ```

5. Deploy and run your application to render the page.

24.6.2 How to Expose the Complete Set of an Extensible Flexfield's Usages, Pages, and Associated Contexts

To expose the complete set of usages, pages, and associated contexts, build an application page comprised of a splitter with two task flows: one containing the list of

extensible flexfield usages and their associated flexfield pages on the left, and the other containing the contexts associated with the selected flexfield page on the right.

To build the application 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 page.

24.6.2.1 Creating the Task Flows

The page requires two task flows: one containing the list of extensible flexfield usages and their associated flexfield pages, and the other containing the contexts associated with the selected flexfield page.

Before you begin:

Create and configure business components to support the extensible flexfield as described in Section 24.5, "Defining and Registering 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**: `oracle.apps.fnd.applcore.flex.eff.runtime.EffCategoryPagesBean`
   - **Value**: `#{pageFlowScope.ContainerBean}`
3. Create the context container task flow, for example, `ContextPageTF`.
4. Add a parameter to the task flow similar to the one in Step 2.

24.6.2.2 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 Section 24.6.2.1, "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 Flex-View library to the page list fragment using the parameters listed in Section 24.6.1.1, "Creating a Task Flow for a Single Extensible Flexfield Usage," with the exception that the value for `_eff_containerBean` should be `pageFlowScope.ContainerBean` instead of `pageFlowScope.EffCategoryPagesBean`.

3. Add the `EffContextsPageContainer` seeded task flow from the ViewController Flex-View library to the context page fragment using the parameters described in Section 24.6.1.1, "Creating a Task Flow for a Single Extensible Flexfield Usage," with the exception that the value for `_eff_containerBean` should be `pageFlowScope.ContainerBean` instead of `pageFlowScope.EffCategoryPagesBean`.
4. Drag and drop the page list fragment to the page-list task flow and the context page fragment in the context task flow.

24.6.2.3 Using the Task Flows in the Page

Use the task flows that you just created in Section 24.6.1.1, "Creating a Task Flow for a Single Extensible Flexfield Usage" to add the usage and context lists to the page.

Before you begin:

1. Create the page list task flow and context container task flow as described in Section 24.6.2.1, "Creating the Task Flows."
2. Create the left and right fragment pages `PageListFrag.jsff` and `ContextPageFrag.jsff` as described in Section 24.6.2.2, "Creating the Fragments."

To use the task flows in the page:

1. Open the page and add the splitter.
2. Add the pageList and ContextPage 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. Follow the steps listed in Section 24.6.1.3, "Rendering the Page" to view the user interface.

24.6.3 How to Expose One Extensible Flexfield Page and Its Contexts

You can incorporate an extensible flexfield into an application as a single task flow that presents a set of contexts associated with a specified extensible flexfield page, which will be identified before the application page initializes.

To expose one extensible flexfield page and its contexts, build an application page including a splitter with a single task flow as a dynamic region on the right, containing the contexts associated with a selected flexfield page. This variation does not present an extensible flexfield usage or page list task flow in the user interface.

24.6.4 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 Section 24.5, "Defining and Registering 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 put the EffCatPageListContainer on the fragment using the parameters described in previous sections.
4. Place the task flow on any page on which you want the page list to appear.
5. Create a task flow, for example, `ContextsTF.xml`.
6. Add an empty fragment to this task flow; for example, `ContextsFrag.jsff`.
7. Access the page on which you would like your contexts to appear. This could possibly be the same page as in Step 4 but is likely the second facet of a splitter.
8. Drag and drop `ContextsTF.xml` as a dynamic region onto this page at the correct location.
9. 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.
10. Open the Java file and add the following code snippet, replacing the value for `taskFlowId` with the appropriate string:

```
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;
  }
}
```

11. Open the page definition for the launch page and scroll to the location of the dynamic region binding.
12. Add the following refresh condition setting to the taskFlow tag:

```
RefreshCondition="#{backingBeanScope.ContextsRenderingBean.refreshRegion}"
```

This provides the call back and the conditions for the refresh of the contexts region.

13. Follow the steps listed in Section 24.6.1.3, "Rendering the Page" to view the user interface.

    When you run the page and click on the page link, the call back to `refreshRegion` will look up the contexts task flow ID and enable a refresh of the dynamic region.

24.7 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 Seed Data Loader.

After you complete the registration process described in Section 24.4.1, "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 the role of the implementer. In addition to the registration data, your flexfield seed data might include contexts, segments, and value sets that you have defined for your flexfield.

For information about extracting and loading seed data, see Chapter 55, "Initializing Oracle Fusion Application Data Using the Seed Data Loader".

24.8 Customizing the Extensible Flexfield Runtime Business Component Modeler

Teams can customize the extensible flexfields base business component modeler and user interface modeler to add additional product-specific logic.

24.8.1 How to Customize the Extensible Flexfield Runtime Business Component Modeler

To extend the extensible flexfield runtime business component modeler, override the `EFFBCModelerFactory.getCustomEFFBCModeler` method, shown in Example 24–3, in the custom extensible flexfield runtime business component modeler factory.

Example 24–3 getCustomEFFBCModeler Method

```
/**
   * Implementing teams need to override this method and provide the
   * custom BC Modeler here (e.g. PIMBCModeler).
   * @param namespace the name space
   * @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);
  }
```

An example of the `PIMBCModelerFactory.java` override for this method is shown in Example 24–4.

Example 24–4 PIMBCModelerFactory.java Override

```
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,
                            isInterface);
  }
```

24.9 Customizing the Extensible Flexfield Runtime User Interface Modeler

If the user interface 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 that the modeler generates for the user interface artifacts just before it persists the generated task flows and JSF

fragments. The implementation classes are in the `oracle.apps.fnd.applcore.flex.uimodeler.customizers` package.

To customize an extensible flexfield business component's generated user interface 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.

24.9.1 How to Create the Customizer Wrapper Class

You can extend the default customizer implementation classes from the `oracle.apps.fnd.applcore.flex.uimodeler.customizers` package to customize the following user interface 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

24.9.1.1 Customizing the Context JSF Fragment

To customize the JSF fragment for a single row context, create a wrapper class for the `SingleRowContextRegionCustomizerImpl` implementation class and override the `customizeGeneratedSingleRowContextFragment` method.

To customize the JSF fragment for a multiple row context, create a wrapper class for the `MultiRowContextRegionCustomizerImpl` implementation class and override the `customizeGeneratedMultiRowContextFragment` method.

24.9.1.2 Customizing the Segment Components in the Generated Context Task Flow

To customize which segment components in the generated context task flow are read-only for single and multi-row contexts, create a wrapper class for the `ContextComponentsCustomizerImpl` implementation class and override the `getAtributeComponentReadonlyELExpression` method. This method returns the value the `readOnly` property a given segment component in the context task flow, as shown in Example 24–5.

Example 24–5 Sample getAtributeComponentReadonlyELExpression Method

```
public String getAtributeComponentReadonlyELExpression(
  Transaction tx, ViewDef contextViewDef) {
        String returnData = null;
        ViewDefImpl contextViewDefImpl = (ViewDefImpl) contextViewDef;

        if(contextViewDefImpl.getName().toUpperCase().indexOf("RESOLUTION") >0 ) {
             returnData = "#{pageFlowScope.EFF_PARAM5=='Y'}";
        }
         return returnData;
    }
```

24.9.1.3 Customizing the Page Links in the Generated Links Task Flow

To customize which page links in a generated links task flow are 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.

24.9.1.4 Customizing the Page Task Flow

To customize which context task flows are 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.

24.9.1.5 Customizing the Search Task Flow

To customize the search task flow, create a wrapper class for the `SearchRegionCustomizerImpl` implementation class. Example 24–4 shows the ways in which you can customize the search task flow and the methods to override to perform the customizations.

Table 24–4 Methods to Override to Customize the Search Task Flow

Customization	Method to Override	Notes
Set the list of actions required in the search results table.	`getResultTableActionMenu`	Use the `ResultsTableActionMenu` inner class to define the menu properties and entries.
Set the managed bean for the generated task flow.	`getSearchManagedBeanClass`	The manage bean must extend `oracle.apps.fnd.applcore.flex.eff.search.ui.bean.DefaultEffSearchManagedBean`
Alter properties in the application table component for search results.	`getApplicationsTablePropertiesMap`	The method returns a hashmap of name-value pairs of the properties to be set.
Alter properties in the ADF table component that is in the application table component for search results.	`getADFTablePropertiesMap`	The method returns a hashmap of name-value pairs of the properties to be set, as shown in Example 24–6.
Customize the query panel properties.	`getQueryPanelPropertiesMap`	The method returns a hashmap of name-value pairs of the properties.
Get a handle to the DOM object of the generated search region and its `pageDef`.	`customizeGeneratedSearchTaskflowRegion`	You only should use DOM objects to customize generated artifacts if there is no other way to perform for the customization.
Customize the `readOnly` property for a component in search results table column.	`getResultsTableColumnComponentReadOnlyELExpression`	This method returns a JSP Expression Language (EL) expression.

Table 24–4 (Cont.) Methods to Override to Customize the Search Task Flow

Customization	Method to Override	Notes
Customize the properties on the search results table column for an `attributeDef`.	`getResultsTableColumnPropertiesMap`	The method returns a hashmap of name-value pairs of the properties
Set the task flow data control scope to shared.	`getSearchTaskFlowDataControlScope`	The task flow can be generated with either a SHARED scope or a ISOLATED scope. The default is ISOLATED. Override this method to set the scope to SHARED.
Add custom toolbar components.	`getAdditionalToolbarComponentDefinitions`	Use the `AdditionalToolbarComponentDefinition` inner class to provide metadata for the required component. You can add components of type CHOICELIST, BUTTON, SPACER, and SEPARATOR.
Set the default search criteria on the search page as it is loaded.	`getDefaultSearchCriteriaName`	
Customize the generated search task flow XML.	`customizeGeneratedSearchTaskflow`	This method returns a handle to the DOM object for the generated task flow XML. You only should use DOM objects to customize generated artifacts if there is no other way to perform the customization.

Example 24–6 Sample getADFTablePropertiesMap Method

```
public HashMap getADFTablePropertiesMap()
    {
        HashMap propertiesMap = new HashMap(10);
        propertiesMap.put("filterVisible","false");
        propertiesMap.put("columnStretching","column:description");
        propertiesMap.put("inlineStyle","width:100%;");
        return propertiesMap;
    }
```

24.9.1.6 How to Create a Metadata Provider Implementation

If you have created customizer wrapper classes for a flexfield business component, you must create a wrapper of the default `UIModelerMetadataProviderImpl` implementation 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 24–7. You only need to override 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 customized

the interface search UI, override the `getInterfaceSearchRegionCustomizer` method. Otherwise, override the `getSearchRegionCustomizerClassName`.

- `getContextComponentCustomizerClassName`
- `getContainerPageCustomizerClassName`
- `getPageListCustomizerClassName`
- `getSearchRegionCustomizerClassName`
- `getInterfaceSearchRegionCustomizer`
- `getSingleRowContextRegionCustomizerClassName`
- `getMultiRowContextRegionCustomizerClassName`

Example 24–7 Sample getSearchRegionCustomizerClassName Method

```
public String getSearchRegionCustomizerClassName() {
        return "oracle.apps.myProduct.myApp.items.eff.ItemSearchRegionCustomizer";
    }
```

24.9.1.7 How to Register the Metadata Provider Class for the Business Component

In order 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.

24.10 Testing the Flexfield

After implementing a flexfield, you can define seed or test value sets for the flexfield, and you can create a model that you can use to test it. For more information, see Section 25.1.2, "How to Test Flexfields".

24.11 Accessing Information About Extensible Flexfield Business Components

The consumers of an extensible flexfield might need to programmatically access an extensible flexfield, such as 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.

24.11.1 How to Access Information About Extensible Flexfield Business Components

You can use the APIs in `oracle.apps.fnd.applcore.flex.runtime.util.common.ExtensibleFlexfieldUtil` to get the names of the following generated extensible flexfield Java business objects (JBOs):

- Context entity object: Use `getContextEoName`, which is shown in Example 24–8.
- Context view object: Use `getContextVoName`, which is shown in Example 24–9.
- Context entity association between base and extension entity objects: Use `getCategoryContextAssocName`, which is shown in Example 24–10.

- Context entity/view object attribute for a given segment code: Use `getContextAttributeName`, which is shown in Example 24–11.
- Categories: Use the various APIs shown in Example 24–12.
- Search view object attributes: Use `getSearchVoAttributeNames`, which is shown in Example 24–13.

Example 24–8 Get EFF Context Entity Object Name

```
/**
   * @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 getContextEoName(Long appId, String flexCode,
                                        String flexUsageCode,
                                        DBTransaction txn,
                                        String contextCode,
                                        String tableType,
                                        String effGroup)
```

Example 24–9 Get EFF Context View Object Name

```
/**
   * @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 getContextVoName(Long appId, String flexCode,
                                        String flexUsageCode,
                                        DBTransaction txn,
                                        String contextCode,
                                        String tableType,
                                        String effGroup)
```

Example 24–10 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 24–11 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
   * object / entity object given the segment code.
   * @param segmentCode - segment code for attribute name.
   * @return attribute name
   */
  public static String getContextAttributeName(String segmentCode)
```

Example 24–12 EFF Category APIs

```
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 String getCategoryContextViewLinkNameForDataEntry()
public static void useServiceProvider()
public static String getCategoryVoNameForSearchCategory()
public static String getCategoryVoNameForSearchGeneric()
```

Example 24–13 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)
```

25

Testing and Deploying Flexfields

This chapter discusses how to test your flexfield business components in Oracle Fusion applications using Integrated WebLogic Server (WebLogic Server), how to deploy your flexfield application to a standalone WebLogic Server instance in order to test the full lifecycle, 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:

25.1 Testing Flexfields

Once your flexfields are available for testing, you can generate test business components and use a Metadata Archive (MAR) profile to run the application.

25.1.1 How to Make Flexfields Available for Testing

Before testing a flexfield in your application, you must ensure that the ADF Business Components model underlying the flexfield is complete. All required entities, view links, application modules, and so on must 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.

You can 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:

- Section 22.3.1, "How to Create Descriptive Flexfield Business Components"

- Section 24.5, "Defining and Registering Extensible Flexfield Business Components"
- Section 23.2.4, "How to Create Key Flexfield Business Components"

It is assumed that the flexfield entity usage has been configured with the settings required to create the business components without developer inputs. In this case, only the following information is needed:

- The usage code of the flexfield entity usage to be tested.
- The output path to where the business components for testing should be written.

25.1.2 How to Test Flexfields

You test a flexfield by running the application using a MAR profile. The MAR profile that you use points to the test business component artifacts.

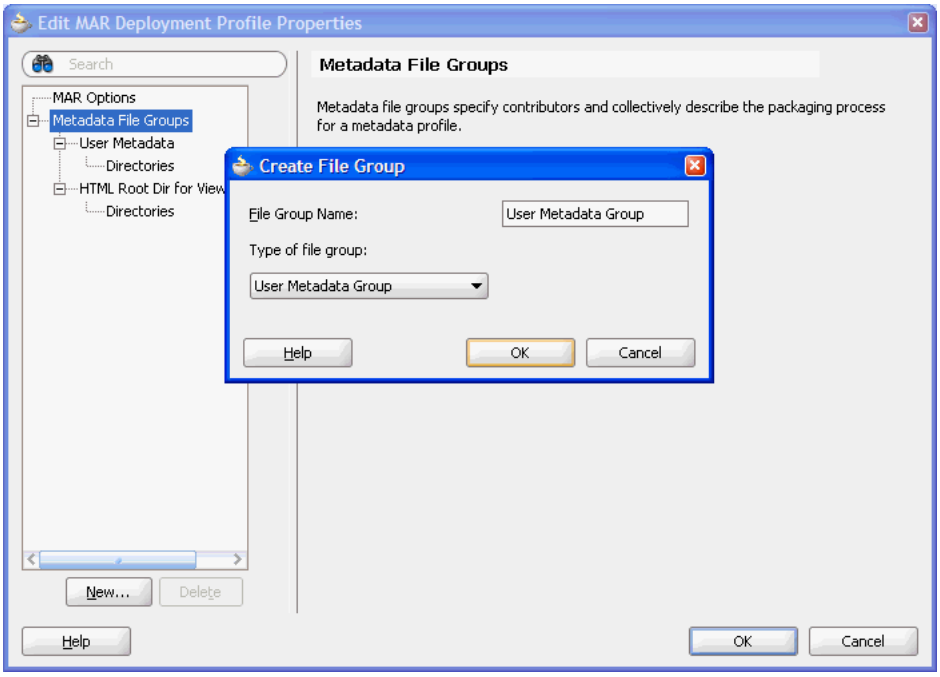
Before you begin:

Create the test business component artifacts for the flexfield, as described in Section 25.1.1, "How to Make Flexfields Available for Testing."

To test a flexfield:

1. Edit your application's properties.
2. Navigate to **Deployment** and click **New** to create a new deployment profile.
3. From the Archive Type dropdown list, select **MAR File** and enter a name for the profile. The Edit MAR Deployment Profile Properties page appears, as shown in Figure 25–1.

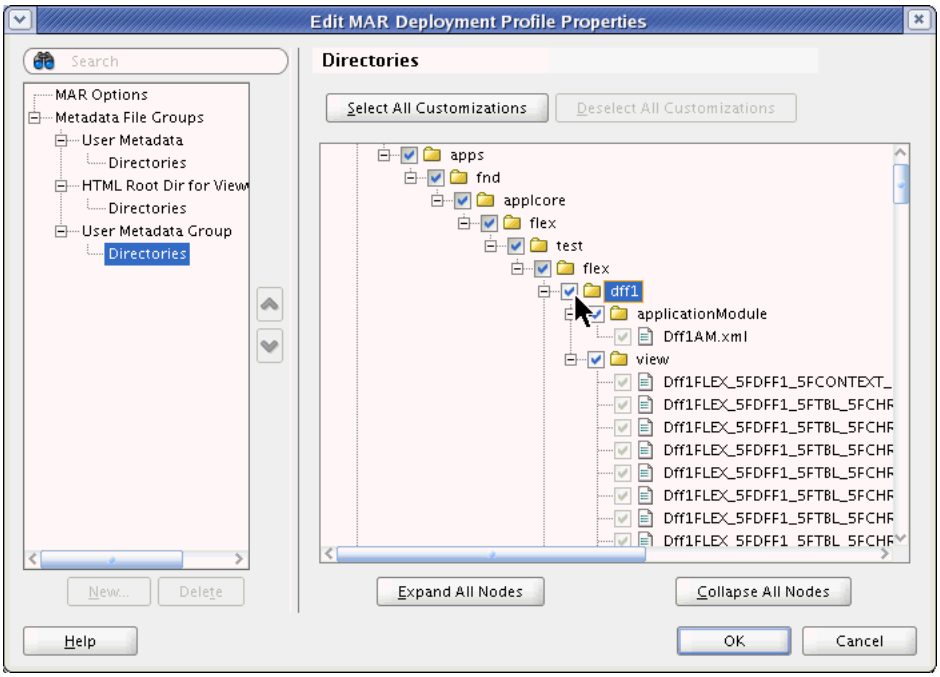
Figure 25–1 New File Group in MAR Profile Properties



4. Navigate to **Metadata File Groups** and click **New** to create a new user metadata group.

5. Enter a name for your user metadata group and click **OK**. The User Metadata Group section appears.
6. Add a contributor, enter the path to your test components (a directory or archive), and click **OK**.
7. Navigate to the Directories section and select the root package of the flexfield to be tested, as shown in Figure 25–2.

Figure 25–2 Root Package of the Flexfield to be Tested



> **Caution:** Be sure to select the correct item. You must click the root package of the flexfield to be tested, so that only the objects below the level of that package come from the test directory. The root package should match the package that you previously registered with your ADF Business Components usage.
>
> As soon as you select the package, 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 Metadata Services (MDS).

8. Edit your application's properties.
9. Navigate to **Run > MDS**, select the MAR profile that you created earlier, and click **OK**.
10. Test your application with the MAR profile.

Note: You will not be able to run application module testers with the test business components because deployment is not needed to run a tester. If you want to run application module testers with the business components that you created for testing, you can create a temporary user library that points to the test components, and include the library in your project as the first library.

25.2 Deploying Flexfields in a Standalone WebLogic Server Environment

Once you have completed the ADF Business Components models underlying the flexfields, and 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, and you have tested your flexfields, you are ready to package and deploy the application to a standalone WebLogic Server environment for full lifecycle testing.

To complete the deployment process for an application that has flexfields:

1. Package the flexfield application.
2. Deploy the application.
3. Configure the flexfields and test the application.

25.2.1 How to Package a Flexfield Application for Deployment

Just as with other Oracle Fusion applications, you must generate an EAR file for deployment to a standalone WebLogic Sever. Before generating the EAR file, you must enable the flexfield packaging plugin.

25.2.1.1 Enabling the Flexfield Packaging Plugin

The flexfield packaging plugin is required to package flexfields from either JDeveloper or the command line. This plugin maps name spaces to the 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
```

25.2.1.2 Generating an EAR File for the Application

To make the flexfield artifacts available at runtime, you package them into the application's application enterprise archive (EAR) file, which subsequently can be installed on the target server.

Before you begin:

Enable the flexfield packaging plugin as described in Section 25.2.1.1, "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 complete, 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 "Deploying Fusion Web Applications" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

- From the command line, run the following `ojdeploy` command:

```
$JDEV_HOME\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 25–1 shows sample tag entries.

Example 25–1 adf-config.xml Flexfield Tag Entries

```
<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.crma.model.view.link.flex;oracle.apps.fnd.applco
re.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>
</mds:metadata-store-usage>
</mds:metadata-store-usages>
</mds:persistence-config>
...
</adf-config>
```

25.2.2 How to Deploy a Flexfield Application

Deploying a flexfield application to a standalone WebLogic Server environment requires additional steps to ensure that all the flexfields' customization metadata is stored in the proper MDS partition.

The process for deploying a flexfield application includes the following tasks:

1. Create an MDS partition to store flexfield customization metadata.
2. Map the product application's EAR file to the MDS partition.
3. Map the Oracle Fusion Middleware Extensions for Applications (ApplCore) Setup application EAR file to the MDS partition.
4. Include the product application model libraries in the ApplCore Setup application.
5. Deploy the product application and the ApplCore Setup application to the WebLogic Server domains.
6. Prime the MDS partition with seeded flexfield artifacts.

25.2.2.1 Creating an MDS Partition

The Metadata Services (MDS) repository is used to store information that enables user customization of 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, you use the WebLogic Scripting Tool (WLST) to create one.

For more information about creating and registering MDS repositories and working with the WebLogic Scripting Tool, see the Managing the Metadata Repository chapter in the *Oracle Fusion Middleware Administrator's Guide*.

Before you begin:

Log into the administration console for the WebLogic Server instance and verify that a JDBC data source exists for the MDS repository. This data source is typically named `mds-ApplicationDBDS`. 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 *Oracle Fusion Middleware Administrator's Guide*.

To Create an MDS Partition:

1. At the command line, type the following line to start the WLST tool.

   ```
   sh $JDEV_HOME/oracle_common/common/bin/wlst.sh
   ```

   On Windows, use `wlst.cmd`.

2. Type 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. Type 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.

```
disconnect()
```

25.2.2.2 Mapping the EAR File to the MDS Partition

To configure the application to store the flexfield customization metadata in the desired MDS partition, you 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 Section 25.2.1.1, "Enabling the Flexfield Packaging Plugin."
- Generate the EAR file as described in Section 25.2.1.2, "Generating an EAR File for the Application."
- Ensure that an MDS partition exists for the flexfield metadata. For more information, see Section 25.2.2.1, "Creating an MDS Partition."
- Obtain the JNDI name for the MDS data source. You can find this value in the **Services > JDBC > Data Sources** page in the WebLogic Server administration console.

To map the EAR file to the MDS partition:

1. At the command line, type the following line to start the WLST tool.

```
sh $JDEV_HOME/oracle_common/common/bin/wlst.sh
```

On Windows, use `wlst.cmd`.

2. Type 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. From the WLST tool, execute the following commands.

```
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`.

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 25–2 shows sample tag entries.

Example 25–2 adf-config.xml metadata-store-usage tags with Added Partition Name

```
<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>
```

25.2.2.3 Mapping the ApplCore Setup Application to the MDS Partition

In order to perform a full lifecycle test, you need to configure the flexfields in the same manner as an implementer would configure them. You use the ApplCore Setup application to complete the flexfield configurations. Just as with the product application's EAR file, you must configure the ApplCore Setup application to store the flexfield customization metadata in the desired MDS partition.

Before you begin:

- Ensure that the `FLEX_DEPLOY_ADDIN_ENABLED` environment variable is set to `true` in your working environment, as described in Section 25.2.1.1, "Enabling the Flexfield Packaging Plugin."
- Ensure that an MDS partition exists for the flexfield metadata. For more information, see Section 25.2.2.1, "Creating an MDS Partition."
- Locate the ApplCore Setup application's `FndSetup.ear` file. This file can typically be found in the `$JDEV_HOME/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 WebLogic Server administration console.

To map the ApplCore Setup EAR file to the MDS partition:

1. At the command line, type the following line to start the WLST tool, if it is not currently running.

   ```
   sh $JDEV_HOME/oracle_common/common/bin/wlst.sh
   ```

   On Windows, use `wlst.cmd`.

2. If you have not yet connected to the server, type 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. From the WLST tool, execute the following commands.

```
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`.

25.2.2.4 Including 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_HOME/jdeveloper/jdev/oaext/external` directory.
2. Execute the following commands to expand the EAR file.

```
mkdir tmpDir
cd tmpDir/
unzip ../FndSetup.ear
```

3. If the `APP-INF/lib` folder does not exist, execute the following commands to create it.

```
mkdir APP-INF
mkdir APP-INF/lib
```

4. Copy all the library JAR files that your product requires to the `APP-INF/lib` folder.
5. Change to the `tmpDir` folder.
6. Execute the following commands to recreate the EAR file with the added JAR files.

```
rm ../FndSetup.ear
zip -r FndSetup.ear .
mv FndSetup.ear ../
rm -Rf tmpDir
```

25.2.2.5 Deploying the Product and Setup Applications to the Server Domains

Once you have mapped the applications as described in Section 25.2.2.2, "Mapping the EAR File to the MDS Partition" and Section 25.2.2.3, "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 Section 25.2.2.4, "Including Product Application Model Libraries in the ApplCore Setup EAR File," you can deploy the applications to the appropriate domains for your topology.

For information about creating domains, see "Creating a WebLogic Domain" in *Oracle Fusion Middleware Creating Domains Using the Configuration Wizard*. For information about installing EAR files, see "Install an Enterprise Application" in the *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Online Help*.

25.2.2.6 Priming 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 into 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 the WLST tool to perform this task.

Before you begin:

Deploy the product and setup applications, as described in Section 25.2.2.5, "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 Section 25.3.2, "How to Use the deployFlexForApp Command."

25.2.3 How to Configure Flexfields

Customers will use the Manage Flexfields tasks to configure the flexfields. For testing purposes, you can use the same tasks in the ApplCore Setup application. For information about using these tasks to configure the flexfields, see the "Using Flexfields for Custom Attributes" chapter in the *Oracle Fusion Applications Extensibility Guide*.

When you have configured a flexfield, click the Deploy button to deploy the configuration to the product application. Because flexfield artifacts are cached per user session, you must log out and log back in to see the deployed configuration.

25.3 Using the WLST Flexfield Commands

You can use the Manage Key Flexfields, Manage Descriptive Flexfields, and Manage Extensible Flexfields tasks to deploy flexfields, as described in the "Using Flexfields for Custom Attributes" chapter in *Oracle Fusion Applications Extensibility Guide*. In addition, the following WebLogic Server Tool commands are available for priming the MDS repository with seeded flexfield artifacts and for deploying flexfields:

- `deployFlexForApp`: Use 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 comment to deploy all flexfields regardless of their status by setting the *force* parameter to `'true'`.
- `deployFlex`: Use to deploy a single flexfield. Deploys the flexfield regardless of status.
- `deleteFlexPatchingLabels`: Use to inquire about or delete all flexfield patching labels.

For information about using the WLST command-line scripting interface, see *Oracle Fusion Middleware Oracle WebLogic Scripting Tool*.

Upon completion, the WLST flexfield commands output a report. As shown in Example 25–3, the report provides the following information for every flexfield that is processed:

- Application ID
- Flexfield code
- Deployment result, such as success or error.

Example 25–3 WLST Flexfield Command Output

```
wls> deployFlexForApp('ApplicationsCoreSetupApp','asdf')
Flexfield Deployment Mbean
name:oracle.apps.fnd.applcore.flex.mbean:name=FlexDeploy,*
<SUCCESS> Connected to Weblogic Server
<SUCCESS> FlexDeploy MBean found :
oracle.apps.fnd.applcore.flex.mbean:name=FlexDeploy,type=AppsRuntimeMBean,Applicat
ion=FndSetup,ApplicationVersion=V2.0
Invoking Mbean : for operation deployFlexForApp
Executed Mbean
*****************FLEXFIELD DEPLOYMENT REPORT******************
The following Flexfields were processed :
1. DescriptiveFlexfield FND_CRM_CASES for ApplicationID 0. Deployment success.
Deployment Log Location :
*****************END******************
```

If errors are encountered, the report provides a deployment error message for the flexfield and lists the usages for which the errors were encountered, as shown in Example 25–4.

Example 25–4 WLST Flexfield Command Output with Errors

```
wls> deployFlex('FLEX_DFF2','DFF')
Flexfield Deployment Mbean
name:oracle.apps.fnd.applcore.flex.mbean:name=FlexDeploy,*
<SUCCESS> Connected to Weblogic Server
<SUCCESS> FlexDeploy MBean found :
oracle.apps.fnd.applcore.flex.mbean:name=FlexDeploy,type=AppsRuntimeMBean,Applicat
ion=FndSetup,ApplicationVersion=V2.0
Invoking Mbean : for operation deployFlex
Executed Mbean
*****************FLEXFIELD DEPLOYMENT REPORT******************
Following DescriptiveFlexfield is processed :
1. DescriptiveFlexfield FLEX_DFF2 for ApplicationID 0. Deployment failed with
ERROR
Flexfield Deployment has failed for 1 entity usages with following exceptions:
[ApplicationId = 0; Flexfield Code = FLEX_DFF2; Flexfield Type = DFF;
Table Usage code :FLEX_DFF2; Table Name :FND_DF_TEST_DFF2_T1;
Eff Group Name :DFF; EntityObject :
oracle.apps.fnd.applcore.flex.test.dff2.model.entity.FndDfTestDff2T1EO:oracle.jbo.
NoDefException: JBO-25002: Definition
oracle.apps.fnd.applcore.flex.test.dff2.model.entity.FndDfTestDff2T1EO of type
Entity Definition is not found.
Deployment Log Location :
************************END*****************************
All Mbeans execution passed.
```

If a runtime exception occurs, the output displays the traceback information.

25.3.1 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 a WebLogic Administration Server for a domain that has a running instance of the ApplCore Setup application. For information on deploying the ApplCore Setup application, see Section 25.2.2.5, "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 WebLogic Administration Server and points to the same database as the ApplicationDB data source.

25.3.2 How 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 Section 25.2.2.6, "Priming the MDS Partition with Configured Flexfield Artifacts."

This command does not deploy flexfields that have a status of DEPLOYED unless the *force* parameter is set to `'true'`.

Before you begin:

1. Configure the product application to store the flexfield customization metadata in the desired MDS partition as described in Section 25.2.2.2, "Mapping the EAR File to the MDS Partition."
2. Map the setup application as described in Section 25.2.2.3, "Mapping the ApplCore Setup Application to the MDS Partition."
3. Deploy the product application and the ApplCore Setup application as described in Section 25.2.2.5, "Deploying the Product and Setup Applications to the Server Domains."
4. Prepare your environment as described in Section 25.3.1, "How to Prepare Your Environment to Use the WLST Flexfield Commands."

To use the deployFlexForApp command:

1. At the command line, type the following line to start the WLST tool, if it is not currently running.

   ```
   sh $JDEV_HOME/oracle_common/common/bin/wlst.sh
   ```

   On Windows, use `wlst.cmd`.

2. 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 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. From the WLST tool, execute the following commands to deploy the artifacts to the MDS partition and to exit the tool.

```
deployFlexForApp('product_application_shortname'[, 'enterprise_id'] [,'force'])
disconnect()
```

Replace *product_application_shortname* with the application's short name wrapped in single-quotes. In a multi-tenant environment, replace *enterprise_id* with the Enterprise ID to which the flexfield is mapped. Otherwise, replace with `'None'` or do not provide a second argument.

To deploy all flexfields regardless of their deployment status, set *force* to `'true'` (the default is `'false'`). If you want to deploy all flexfields in a single-tenant environment, you either can set *enterprise_id* to `'None'`, or you can use the following signature:

```
deployFlexForApp(
applicationShortName='product_application_shortname',force='true')
```

> **Tip:** The application's short name is the same as the application's module name. For more information, see Appendix A, "Working with the Application Taxonomy."

4. Optionally, log into the application and view the pages that contain flexfields. If you have seeded any flexfield configurations by defining value sets, segments, context, 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.

25.3.3 How to Use the deployFlex Command

The `deployFlex` command deploys the specified flexfield to the specified product application. Deploys the flexfield regardless of status.

Before you begin:

Prepare your environment as described in Section 25.3.1, "How to Prepare Your Environment to Use the WLST Flexfield Commands."

To use the deployFlex command:

1. At the command line, type the following line to start the WLST tool, if it is not currently running.

```
sh $JDEV_HOME/oracle_common/common/bin/wlst.sh
```

On Windows, use `wlst.cmd`.

2. 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 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. From the WLST tool, execute the following commands to deploy the artifacts to the MDS partition and to exit the tool.

```
deployFlex('flex_code', 'flex_type')
```

```
disconnect()
```

Set the command parameters as follows:

- Replace *flex_code* with the code that identifies the flexfield
- Replace *flex_type* with the flexfield's type, which is either `DFF`, `KFF`, or `EFF`.

The values must be wrapped in single-quotes.

4. Optionally, log into the application and view the pages that contain the flexfield. If you have seeded the flexfield's configurations by defining a value set, segments, context, or structures, for example, the flexfield should appear on the appropriate pages. If the flexfield has not been configured, the corresponding user interface sections will be blank.

25.3.4 How to Use the deleteFlexPatchingLabels Command

Whenever a `deployPatchedFlex()` WLST command is used to deploy flexfield changes to MDS, an MDS label is created in the format `FlexPatchingWatermark`*date+time*. You use the `deleteFlexPatchingLabels` command to inquire about and delete these labels.

Before you begin:

Prepare your environment as described in Section 25.3.1, "How to Prepare Your Environment to Use the WLST Flexfield Commands."

To use the deleteFlexPatchingLabels command:

1. At the command line, type the following line to start the WLST tool, if it is not currently running.

   ```
   sh $JDEV_HOME/oracle_common/common/bin/wlst.sh
   ```

   On Windows, use `wlst.cmd`.

2. 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 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. From the WLST tool, execute the command in the appropriate format depending upon whether you want to inquire about labels or delete labels.

   - Execute the command with the `infoOnly` argument to output a list of flexfield patching labels.

     ```
     deleteFlexPatchingLabels(infoOnly='true')
     ```

     The command outputs a list of patching labels similar to the following example.

Example 25–5 deleteFlexPatchingLabels Information Only Output

```
Found Flex Metadata label FlexPatchingWatermark2011-08-12 05:50:48
Found Flex Metadata label FlexPatchingWatermark2011-08-12 12:32:59
Found Flex Metadata label FlexPatchingWatermark2011-08-12 12:28:19
```

- Execute the command with no arguments to delete the flexfield patching labels.

```
deleteFlexPatchingLabels()
```

4. Execute the following command to disconnect.

```
disconnect()
```

25.4 Regenerating Flexfield Business Components Programmatically

After you complete the flexfield development activities to incorporate a flexfield into your application, you might need to update the descriptive flexfield implementation in your application at a later time by repeating the process of creating the flexfield business components.

Rather than recreate the flexfield business components manually using the Create Flexfield Business Components wizard, you can instead invoke the flexfield business component modeler programmatically using a Java program. In this way the creation of the business components can be automated without user interaction.

The flexfield business component modeler can be invoked trough the Java API only in a deployed web application. You must have the following:

- 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 can either 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 25–6 demonstrates appropriate Java code for updating the business components for a descriptive flexfield.

Example 25–6 Java Code for Invoking the Flexfield Business Component Modeler

```
import oracle.apps.fnd.applcore.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@dev1.us.oracle.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.applcore.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();
    }
  }
}
```

Examine `BCModeler.Option` for usage-specific settings.

25.5 Integrating Flexfield 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. For example, an HR application can register setup activities such as "Create Employees" and "Manage Employee Tree Structure." Implementers and administrators use these registered task flows, which are accessed from the Fusion Applications Setup and Maintenance work area, 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 Chapter 24.4, "Defining and Registering Extensible Flexfields." You must use the `FND_FLEX_KF_SETUP_APIS` PL/SQL API to register key flexfields as described in Section 23.2.1, "How to Develop Key Flexfields."

Table 25–1 lists the flexfield setup task flows. To make these task flows available to developers, implementers, or administrators, you register the appropriate task. For more information, see the *Oracle Fusion Applications Common Implementation Guide*.

For information about using the tasks for managing the flexfields, see the "Using Flexfields for Custom Attributes" chapter in the *Oracle Fusion Applications Extensibility Guide*. For more information about the Register Descriptive Flexfields task, see Section 22.2.2, "How to Register and Define Descriptive Flexfields."

Table 25–1 Flexfields Task Flows and Parameters

Task Flow Name	Task Flow XML	Parameters Passed	Behavior

Register Descriptive Flexfields	`/WEB-INF/oracle/apps/fnd/applcore/flex/dff/ui/publicFlow/RegisterDescriptiveFlexfieldsTF.xml#RegisterDescriptiveFlexfieldsTF`	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 registration metadata.

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 25–1 (Cont.) Flexfields Task Flows and Parameters

Task Flow Name	Task Flow XML	Parameters Passed	Behavior
Manage Extensible Flexfields	/WEB-INF/oracle/apps/fnd/applcore/flex/dff/ui/publicFlow/ManageExtensibleFlexfieldsTF.xml#ManageExtensibleFlexfieldsTF	To invoke search mode: mode='search' To restrict search mode to extensible flexfields belonging to a particular product module: mode='search' moduleType='*moduletype*' moduleKey='*modulekey*' To invoke edit mode for a specific extensible flexfield: mode='edit' extensibleFlexfieldCode=*effcode* applicationId=*appid* To optionally specify a page heading for the task flow: pageTitle='*titlestring*'	Search and edit extensible flexfield configuration.
Manage Key Flexfields	/WEB-INF/oracle/apps/fnd/applcore/flex/kff/ui/publicFlow/ManageKeyFlexfieldsTF.xml#ManageKeyFlexfieldsTF	To invoke search mode: mode='search' To restrict search mode to key flexfields belonging to a particular product module: mode='search' moduleType='*moduletype*' moduleKey='*modulekey*' To invoke edit mode for a specific key flexfield: mode='edit' keyFlexfieldCode=*kffcode* applicationId=*appid* To optionally specify a page heading for the task flow: pageTitle='*titlestring*'	Search and edit key flexfield configuration.
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 Chapter 49, "Implementing Function Security".

Part V

Using Oracle Enterprise Crawl and Search Framework

This part of the Developer's Guide provides information about the Oracle Enterprise Crawl and Search Framework (ECSF). *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.

The *Getting Started with Oracle Enterprise Crawl and Search Framework* chapter discusses how to set up ECSF.

The *Creating Searchable Objects* chapter discusses how to create sets of data that make view objects available for full text search.

The *Configuring ECSF Security* chapter discusses how to configure security for ECSF.

The *Validating and Testing Search Metadata* chapter discusses how to validate and test the search metadata.

The *Deploying and Crawling Searchable Objects* chapter discusses how to deploy the sets of data to the ECSF application and verify the crawl.

The *Advanced Topics for ECSF* chapter discusses the additional functionality that ECSF offers to enhance the search experience.

This part contains the following chapters:

26

Getting Started with Oracle Enterprise Crawl and Search Framework

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:

26.1 Introduction to Using Oracle Enterprise Crawl and Search Framework

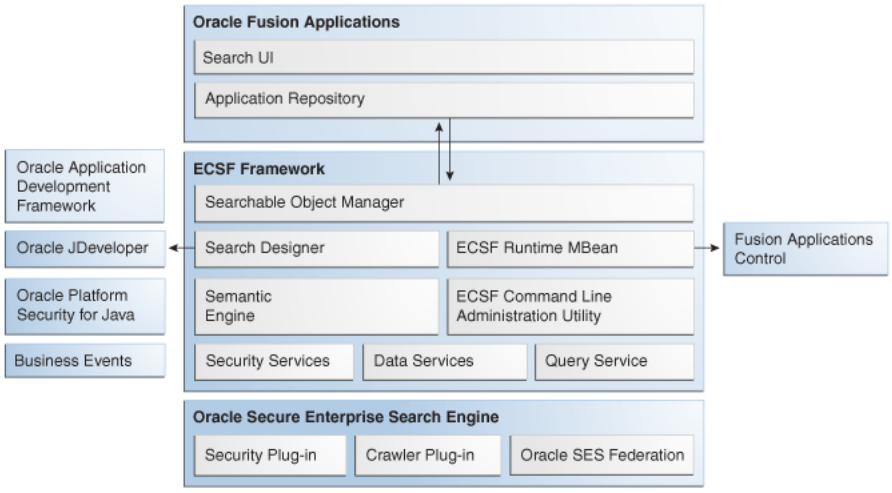
Oracle Enterprise Crawl and Search Framework (ECSF) is an Oracle Fusion Middleware search framework that enables you to quickly expose application context information on various business objects to enable full-text transactional search.

For more information, see the "Managing Search with Oracle Enterprise Crawl and Search Framework" chapter in the *Oracle Fusion Applications Administrator's Guide*.

26.1.1 ECSF Architecture

The ECSF framework 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 26–1 illustrates the architecture of the ECSF framework.

Figure 26–1 Enterprise Crawl and Search Framework Architecture



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.

26.1.1.1 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.

26.1.1.2 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.

26.1.1.3 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

26.1.1.4 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, as well as for administering the runtime parameters of ECSF, the target search engine, and the configuration of parameters.

26.1.1.5 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.

26.1.1.6 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 Chapter 28, "Configuring ECSF Security".

26.1.1.7 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.

26.1.1.8 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.

26.1.1.9 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.

For more information about Oracle SES, see *Oracle Secure Enterprise Search Administrator's Guide*.

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.

26.1.1.10 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 in order 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 Chapter 28, "Configuring ECSF Security".

26.1.1.11 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.

26.2 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.

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. In order 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 *Oracle Fusion Applications Administrator's Guide*.
6. (Optional) Configure the log settings.
7. (Optional) Set the startup parameter to support taking input from a text file.

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.

26.2.1 How to Make 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:

- `/net/mount1/appbase/fusionapps/applications/fscm/deploy/EarFscmSearch.ear/APP-INF/lib/`*`searchable_object_jar_file`*
- `/net/mount1/appbase/fusionapps/applications/crm/deploy/EarCrmSearch.ear/APP-INF/lib/`*`searchable_object_jar_file`*
- `/net/mount1/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. For information, see the "Adding ADF Library Components into Projects" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

The application JAR file, which contains the searchable objects that are defined in your application, is written to the `deploy` directory of the project.

In order 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 Section 26.2.2, "How to Set the Class Path".

If you are deploying searchable objects from multiple applications, you must create a JAR file for each of those applications in order to add the searchable objects to the class path.

26.2.2 How to Set 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.

26.2.2.1 Setting 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:

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 Section 26.2.1, "How to Make 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.

26.2.2.2 Setting 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:

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/view_storage/login_fmwtools_view/fmwtools/mw_home/oracle_common"`.
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 Section 26.2.1, "How to Make 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.

26.2.3 How to Set 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. In order 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 Section 26.2.4, "How to Manually Connect to the Oracle Fusion Applications 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.

26.2.3.1 Setting 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:

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=(PROTO-
     COL=tcp)(HOST=fusionhost123)(PORT=5521))(CONNECT_
     DATA=(SID=dbmsdb2)))'
     ```

     – Using service name:

```
set CONNECT_INFO=connect to database descriptor
'(DESCRIPTION=(ADDRESS=(PROTO-
COL=tcp)(HOST=fusionhost123)(PORT=5521))(CONNECT_
DATA=(SERVICE NAME=myservice)))'
```

3. Save the script file.

26.2.3.2 Setting 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:

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))(C
     ONNECT_DATA=(SID=dbmsdb2)))'
     ```

     – Using service name:

     ```
     set CONNECT_INFO=connect to database descriptor
     '(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=fusionhost123)(PORT=5521))(C
     ONNECT_DATA=(SERVICE NAME=myservice)))'
     ```

3. Save the script file.

26.2.4 How to Manually Connect 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. In order 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 Section 26.2.3, "How to Set 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 the connection to the Oracle Fusion Applications database:

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))(CONNE
  CT_DATA=(SID=dbmsdb2)))'
  ```

  – Using service name:

  ```
  '(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=fusionhost123)(PORT=5521))(CONNE
  CT_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))(CONNE
  CT_DATA=(SID=dbmsdb2)))'
  ```

  – Using service name:

  ```
  '(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=fusionhost123)(PORT=5521))(CONNE
  CT_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.

26.2.5 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:

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.

26.2.6 How to Configure 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/`.

26.2.7 How to Automate 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 are typically prompted for (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 in order to exit the ECSF Command Line Administration Utility. Example 26–1 illustrates a sample list of commands for an input file.

Example 26–1 Sample Input File

```
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
```

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.

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.

To set the JPS Config file path:

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,

   ```
   STARTUP_PARAMS="AUTOMATE /scratch/commands.txt"
   ```

   where `commands.txt` is the input filename.
3. 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.

26.3 Setting Up Oracle Enterprise Manager and Discovering ECSF

While the ECSF Command Line Administration Utility can be used to quickly test and manage the searchable objects, Oracle Enterprise Manager Fusion Applications Control should be used 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.

26.3.1 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:

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.

26.3.1.1 Adding 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:

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`:

```
<listener>
<listener-class>oracle.ecsf.mbean.RegisterMbeanContextListene
r</listener-class>
</listener>
```

3. Save.

26.3.1.2 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.

26.3.1.3 Configuring Data Sources in Oracle WebLogic Server

You must configure data sources in Oracle WebLogic Server for MBean integration. For information, see *Oracle Fusion Middleware Configuring Server Environments for Oracle WebLogic Server*.

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.

26.3.1.4 Deploying the ECSF Application Using the EAR File

Make the MBean available by deploying the EAR file you created to Integrated WebLogic Server. For information, see *Oracle Fusion Middleware Configuring Server Environments for Oracle WebLogic Server*.

Make sure that the EAR file is deployed and the application status is active in the final step.

26.3.1.5 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.

26.3.2 How to Install Oracle Enterprise Manager

You must install Oracle Enterprise Manager in order to access the Fusion Applications Control. Installing Enteprise Manager allows you to then enable it to discover the ECSF custom application target in Oracle WebLogic Server.

26.3.3 How to Discover ECSF in Oracle Enterprise Manager

In order 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. Once 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:

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. Once it is discovered, you can directly launch EM with the following URL:
>
> `http://`*`your machine name`*`:`*`port`*`/em`

26.3.4 How to Add Users to the Administrators Group

In order 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. For information, see *Oracle Fusion Middleware Securing Oracle WebLogic Server*.

27

Creating Searchable Objects

This chapter describes how to create searchable objects in Oracle Fusion Applications.

This chapter includes the following sections:

27.1 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. Since the binding information describes how a searchable object behaves in a given context, it is sometimes called search metadata.

To create searchable object, 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 Section 31.2, "Enabling Search on Fusion File Attachments".
3. (Optional) Enable the capability to crawl searchable objects with Oracle WebCenter tags. For more information, see Section 31.3, "Enabling Search on WebCenter Tags".
4. Secure the searchable objects
5. Configure the search features
6. Configure the custom properties for the searchable objects

27.2 Defining Searchable Objects

Oracle Enterprise Crawl and Search Framework 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 its attributes for search and creates the necessary metadata for ECSF. The ECSF metadata can be packaged into an application archive and subsequently be used by ECSF runtime to deploy data sources into Oracle Secure Enterprise Search (Oracle SES) to perform crawling 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 in order to make the searchable objects usable during runtime for both crawl and query.

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, only include attributes you intend to search).
- Exclude locator objects (LOB) from searchable objects unless you intend to search LOB contents.
- Restrict the use of multilevel view objects to 5 levels to avoid causing severe feed performance degradation.
- Co-locate 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.
- 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.
- Attachments (Oracle Content Management or LOBs) require record by record processing, and every record requires one additional network round-trip.
- Store attributes only when necessary. Attributes stored in Oracle SES should be for only the following purposes:
  - Faceted navigation
  - Advanced search
  - Search result actions
  - Primary keys

  If the value of an attribute needs to be searchable, 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 Section 27.2.10, "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.
- Determine 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 are 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 Section 27.2.6.1, "Making View Object Attributes Searchable".

  – Raising change events. For information, see Section 31.8, "Raising Change Events Synchronously".

  If neither mechanism is implemented, the data that has been added or modified since the initial crawl will not be indexed, and therefore will not be searchable.

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.

Note: You must choose the **Oracle Fusion Applications Customization** role when you launch JDeveloper.

You can also switch to the **Oracle Fusion Applications Customization** role from within JDeveloper by selecting **Tools > Preferences > Roles** from the main menu and restarting JDeveloper.

Before you begin:

Before you can define searchable objects, you must be familiar with the following:

- Oracle JDeveloper and Oracle Application Development Framework technology, and understand how to build applications using entity objects, view objects, and view links. For information, see *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
- How view objects are used in ECSF. For more information, see the "Managing Search with Oracle Enterprise Crawl and Search Framework" chapter in the *Oracle Fusion Applications Administrator's Guide*.
- How to use Groovy expressions when defining search metadata. For information, see Section 27.2.1, "How to Use Groovy Expressions in ECSF".

27.2.1 How to Use 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 leverages Oracle Application Development Framework's (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. Once 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.

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.

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 accessors in either end of the view link. Only generate the view link accessor 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 Section 31.10, "Localizing ECSF Artifacts".

27.2.1.1 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 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 make sure to 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 Section 27.2.9, "What You May Need to Know about Preventing Conflicts with Oracle SES Default Search Attributes".

27.2.1.2 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 title value:

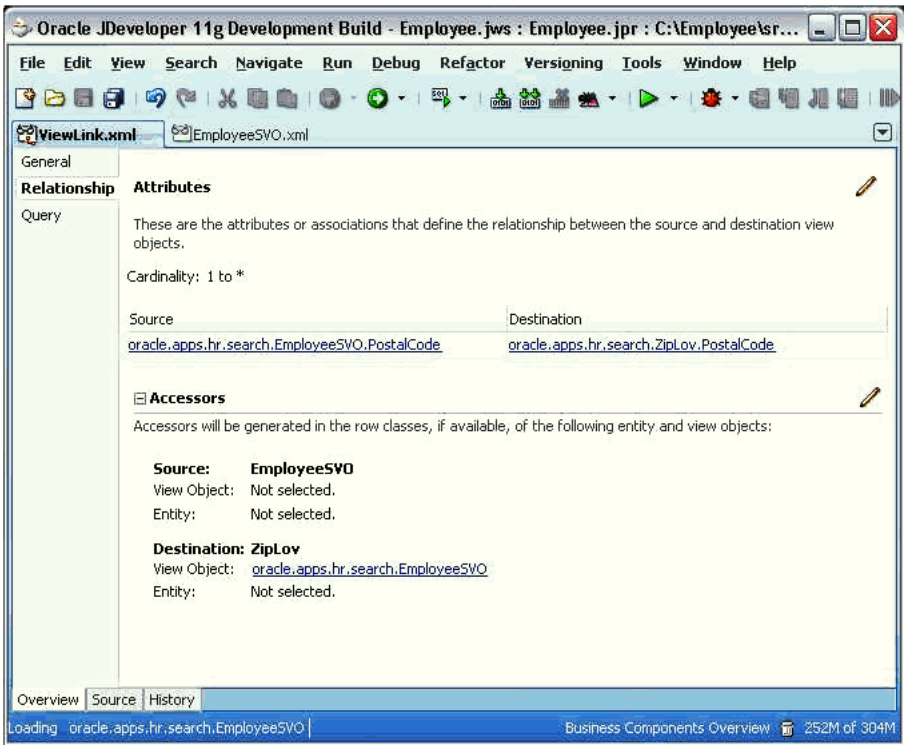
```
Product Codes: 1XYZ
```

If a child view object referenced by an expression contains more than one record, then the values of all the child records are concatenated together. 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 27–1, which shows the accessors information on the **Relationship** navigation tab of the view link editor, the name of the view link accessor is `ZipLov`.

Figure 27–1 Location of View Link Accessor Name for Child View Object



You can also find the name of view link accessors in the `<ViewLinkAccessor>` tags of the XML source mode of the view object.

Alternatively, you can use the `curdoc` keyword to access the current document, as shown in Example 27–1.

Example 27–1 Using curdoc to Access Child Documents and Their Attributes

```
s = "";
if (curdoc.getChildDocs("ProductsView") != null)
{
  for (d in curdoc.getChildDocs("ProductsView"))
  {
    s = s + d.attributes.ProdCode + " ";
  }
};
"Product Codes: " + s;
```

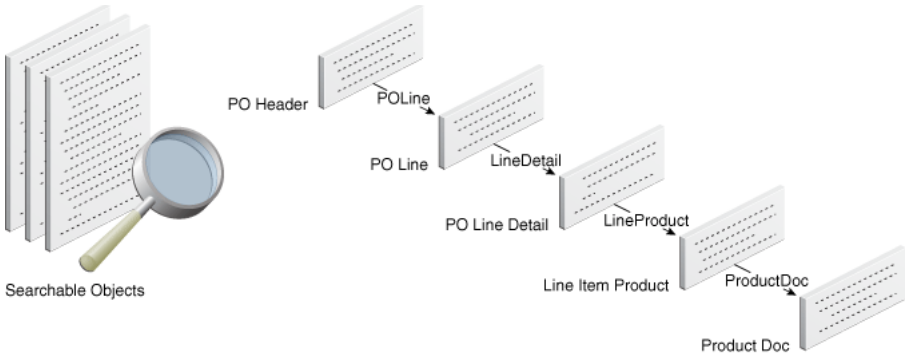
The `curdoc` keyword is used to access child documents and their attributes.

27.2.1.3 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 27–2 illustrates a searchable object with a 5 level structure.

Figure 27–2 Searchable Object with 5 Levels



The figure shows a representation of the `PurchaseOrder` searchable object with 5 levels.

Using Groovy expressions in the search metadata, you can access attributes in `PO Line Detail`, `Line Item Product`, and `Product Doc`. For example, you can enter the following Groovy expression in the **Title** field:

```
POLine.LineDetail.AttributeX +
POLine.LineDetail.LineProduct.AttributeY +
POLine.LineDetail.LineProduct.ProductDoc.AttributeZ
```

If a grandchild view object referenced by an expression contains more than one record, then the values of all the grandchild records are concatenated together. For example, if you reference view object attributes with *`viewLinkAccessorName.viewLinkAccessorName.voAttrName`*, the result of the expression `POLine.LineDetail.AttributeX` is the concatenation of all `AttributeX` values for the `LineDetail` of all `POLines` of the current `PurchaseOrder`.

27.2.1.4 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 27–2, in the **Body** field.

Example 27–2 Sample Groovy Expression to Determine Attribute's Java Data Type

```
'Hiredate type: ' + (Hiredate != null ? Hiredate.getClass().getName() : 'null');
```

The class name of the `Hiredate` attribute's value is displayed and can be viewed in the Data Feed:

```
Hiredate type: java.sql.Date.
```

Once you determine the Java data type of a view object attribute, you can apply standard Java formatting techniques to format its value. Example 27–3 shows a sample Groovy expression for formatting a date.

Example 27–3 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);
```

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 27–4.

Example 27–4 Sample Groovy Expression for Formatting Numbers

```
fmt = new java.text.DecimalFormat('#,###,###');
'Quantity: ' + fmt.format(Qty);
```

The Groovy expression evaluates to:

```
Quantity: 2,450
```

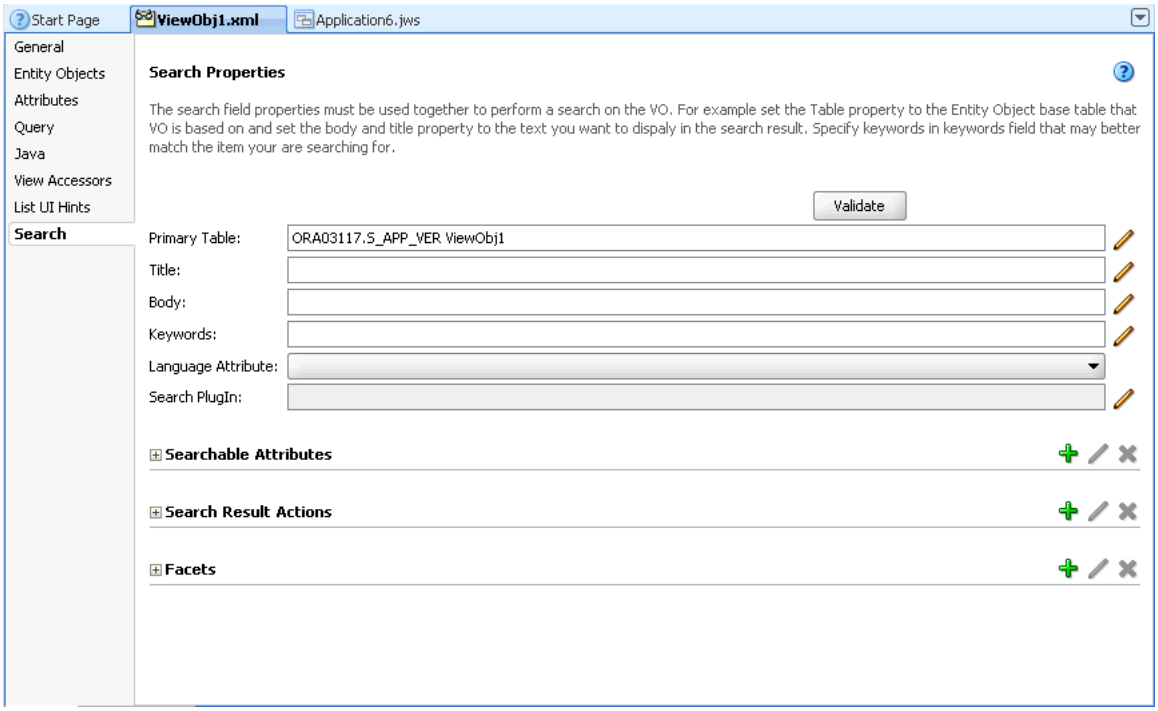
27.2.2 What Happens When You Use Groovy Expressions in ECSF

The Groovy expressions are compiled and evaluated at runtime to display the desired string value in the crawl data feeds.

27.2.3 How to Make View Objects Searchable

Use the Search page of the overview editor in JDeveloper, as shown in Figure 27–3, to set search property values for view objects.

Figure 27–3 Search Page for Configuring Search Properties of View Objects



Note: The Search page is not editable if the view object or existing searchable object is read-only.

27.2.3.1 Setting Search Property Values for View Objects

Make view objects searchable by setting search property values for them.

Caution: Do not modify the Oracle Fusion Applications Help searchable object named `TopicSearchPVO`.

To set search property values for view objects:

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 type directly in the text box or use the Select Primary Table dialog. For more information, see Section 27.2.3.2, "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 Section 27.2.1, "How to Use 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 on the search results. This appears below the title. For more information, see Section 27.2.1, "How to Use 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 Section 27.2.1, "How to Use 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, and sent to Oracle SES as part of the document for indexing. Once 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, leave it blank.

   > **Note:** Currently, ECSF provides a way for you to specify whether 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, 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 Section 27.2.3.3, "Using the Search PlugIn Dialog".

8. Save the view object.

Note: In order 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`), you must close and reopen the view object to view the updates.

27.2.3.2 Using the Select Primary Table Dialog

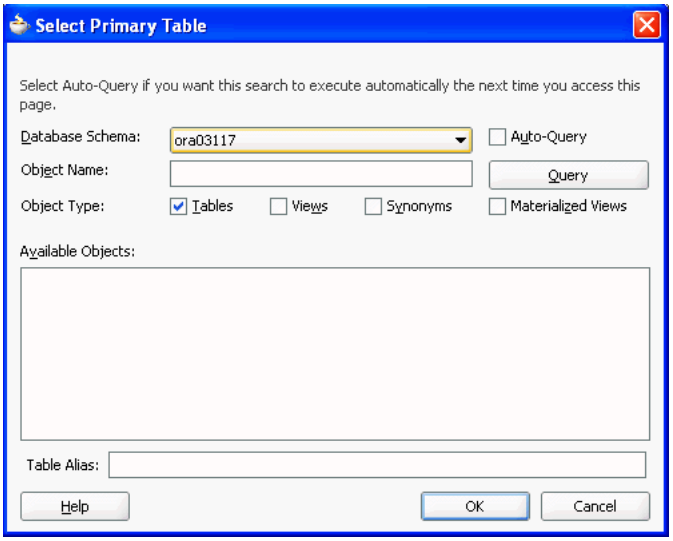
Use the Select Primary Table dialog to specify the primary table for the searchable object if the view object is not based on an entity object.

To use the Select Primary Table dialog

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 27–4.

Figure 27–4 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. Type 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 in order to save.

27.2.3.3 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 wish 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 Section 27.2.6.1, "Making View Object Attributes Searchable".

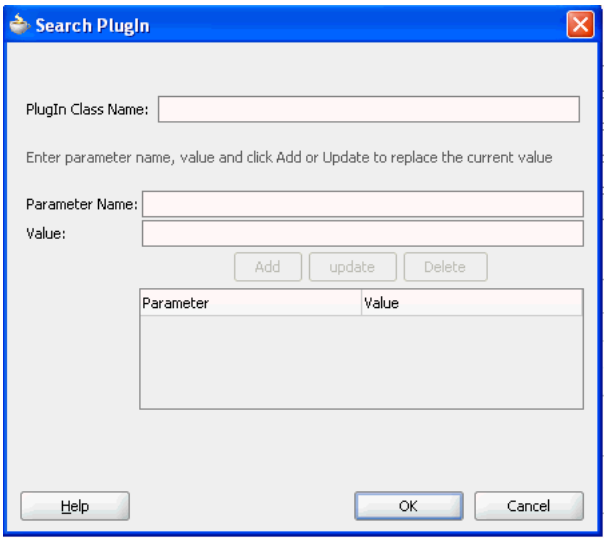
Note: If the search extension is in the Oracle WebLogic Server shared library, then the ECSF library (`ecsf.jar`) must be present in the shared library in order for ECSF to load the interface.

To use the Search PlugIn dialog:

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 27–5.

Figure 27–5 Search PlugIn Dialog



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. If desired, enter a parameter name in the **Parameter Name** field and its corresponding value in the **Value** field, then click **Add**.
4. To update an existing parameter, select the desired parameter in the table, change its value, then click **Update**.
5. To delete an existing parameter, select the desired parameter in the table and click **Delete**.
6. Click **OK** to save.

27.2.4 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, 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.

27.2.5 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.

27.2.6 How to Make View Object Attributes Searchable

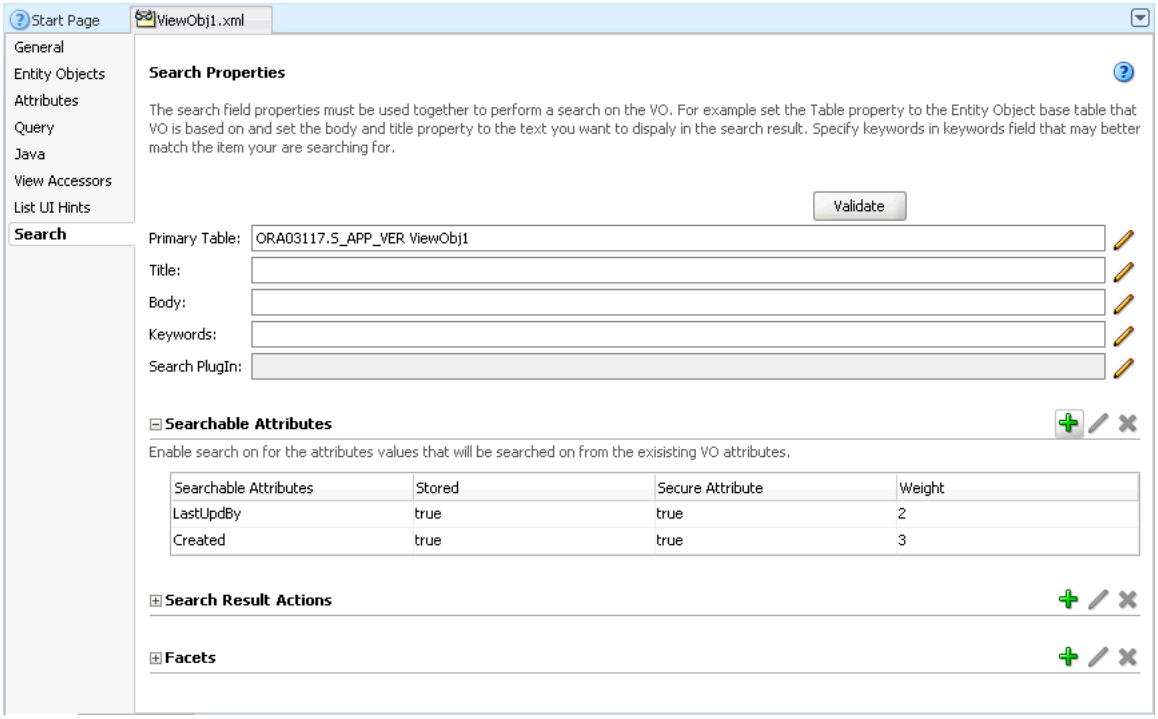
Making view object attributes searchable creates the necessary metadata for:

- 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 page of the overview editor in JDeveloper, shown in Figure 27–6, to set property values for view object attributes.

Figure 27–6 Search Page for Configuring Search Properties of View Object Attributes



Using the Search page, you can perform the following tasks on view object attributes:

- Make view object attributes searchable.
- Modify searchable attributes.
- Delete searchable attributes.

27.2.6.1 Making View Object Attributes Searchable

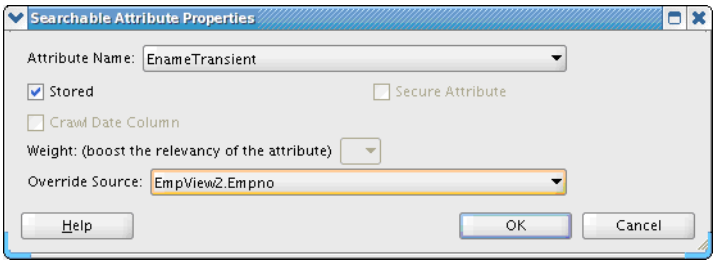
Make view object attributes searchable by setting search property values for them.

To set search property values for view object attributes:

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 27–7.

Figure 27–7 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 27–1.

> **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 27–1 Searchable Attribute Properties

Property	Description
`Stored`	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 Section 27.2.10, "What You May Need to Know about Preventing Search Attribute Naming Conflicts." Attributes stored in Oracle SES are used for Advanced Search, Actionable Results, and Faceted Navigation. For more information, see Section 27.4.1, "How to Define Search Result Actions"and Section 27.4.4, "How to Implement Faceted Navigation". Selecting this checkbox enables the **Weight** field.
`Secure Attribute`	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.

Table 27–1 (Cont.) Searchable Attribute Properties

Property	Description
`Crawl Date Column`	These are date columns that are checked 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** in a SQL predicate for ECSF to detect added or changed data. During incremental crawls, scheduled using either the 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 **Crawl Date Column** 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.
`Weight`	This field is enabled only when you select the **Stored** checkbox. 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, the stored attribute gets no boost. If the weight is set to 2 or 3, the stored attribute gets very low boost (added to custom attribute Headline2). If the weight is set to 4 or 5, the stored attribute gets low boost (added to custom attribute Headline1). If the weight is set to 6 or higher, the stored attribute gets high boost (added to custom attribute Reference Text).
`Override Source`	The Override Source property supports allowing facets to be based on child VO attributes. This feature lets a Stored Attribute be defined against a child VO attribute and for a facet to be defined against this stored attribute. This allows users to filter search results such that only results whose child VO attribute matches a certain value will be returned. The Override Source property will support values of the format `ViewLinkAccessorName.AttributeName`

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), you must close and reopen the view object to view the updates.

27.2.6.2 Modifying Searchable Attributes

Modify a searchable attribute by editing the search property values for the view object attribute.

To edit search property values for view object attributes:

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 27–7.

   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 27–1.
6. Click **OK**.
7. Save the view object.

27.2.6.3 Deleting Searchable Attributes

Deleting the searchable attributes removes the search metadata for the view object attribute.

To delete searchable attributes:

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.

27.2.7 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 search 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, 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 tags are associated with the searchable object. For information, see Section 31.3, "Enabling Search on WebCenter Tags".
- `DEFAULT_ACL_KEY`. ECSF uses this attribute to store access control list (ACL) keys for the document.

27.2.8 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:** Manually deleting an attribute that has a corresponding search attribute from a view object causes unexpected search results.

27.2.9 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.

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, 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 won't 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.

27.2.10 What You May Need to Know about Preventing Search Attribute Naming Conflicts

Because Oracle SES only supports facets on attributes of type STRING, if a facet is defined on a non-string attribute 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. See Section 27.2.10.1, "Checking for Stored Attribute Conflicts." Take the following example:

You create a view object oracle.apps.crm.cutomer360.CustomerPVO 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:

Attribute Name	Attribute Alias	Attribute Type
Name	NAME	VARCHAR2
OrganizationId	ORGANIZATION_ID	NUMBER
Description	DESCRIPTION	VARCHAR2

You use the Search Designer to annotate a subset of these attributes (Name and OrganizationId) to store in Oracle SES for search purposes.

When Oracle SES crawls oracle.apps.crm.cutomer360.CustomerPVO, ECSF sends a document for each customer in the table. Each document contains the following attribute details:

Attribute Alias	Attribute Type
NAME	VARCHAR2
ORGANIZATION_ID	NUMBER

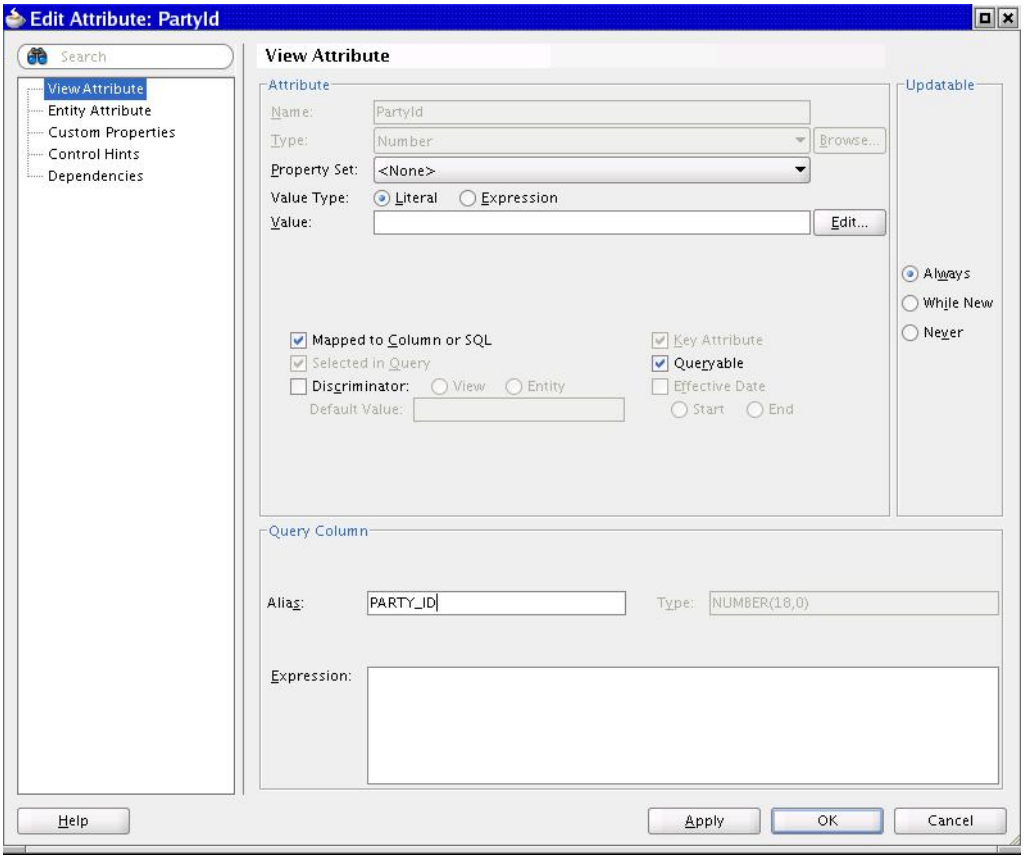
In Oracle SES, NAME and ORGANIZATION_ID are created as customer attributes with the respective types. Due to the global nature of custom attributes in Oracle SES, if an attribute of the same name already exists, no new attribute is created even if its type is different. Values for the attributes with 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 if an attribute is used for a facet or not. If it is used for a facet, 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. Navigate to the JDeveloper ADF view object attribute editor and update the Alias property value, as shown in Figure 27–8. For example, in the view object attribute editor, change the Alias value from PARTY_ID to PARTY_ID_FACET.

Figure 27–8 Changing the Alias Value in the Attribute Editor



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*.

27.2.10.1 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.

27.3 Securing Searchable Objects

ECSF determines if a user has access to a search category depending on whether or not 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

27.3.1 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 Section 27.2.3.3, "Using the Search PlugIn Dialog".

To set permissions for searchable objects:

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.

27.3.2 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:

1. Expand **Application Resources > Descriptors > META-INF**, and open the `jazn-data.xml` file.
2. Select **Identity Store**, and click **New** to add a new realm. Name it `jazn.com`.
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 27–5.

Example 27–5 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>
```

   The user `Scott` is associated to the `jazn.com` security realm.

4. Save.

27.3.3 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 an application policy store:

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_VIEW_DETAILS`

   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 27–6.

Example 27–6 Application Policy Store

```
<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>
               </grantee>
               <permissions>
                  <permission>
                     <class>oracle.adf.share.security.authorization.RegionPermission</class>
                     <name>PURCHASE_ORDER_VIEW_DETAILS</name>
                     <actions>view</actions>
                  </permission>
               </permissions>
            </grant>
         </jazn-policy>
```

```
      </application>
   </applications>
</policy-store>
```

The application policy store includes the values that you specified.

27.4 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.

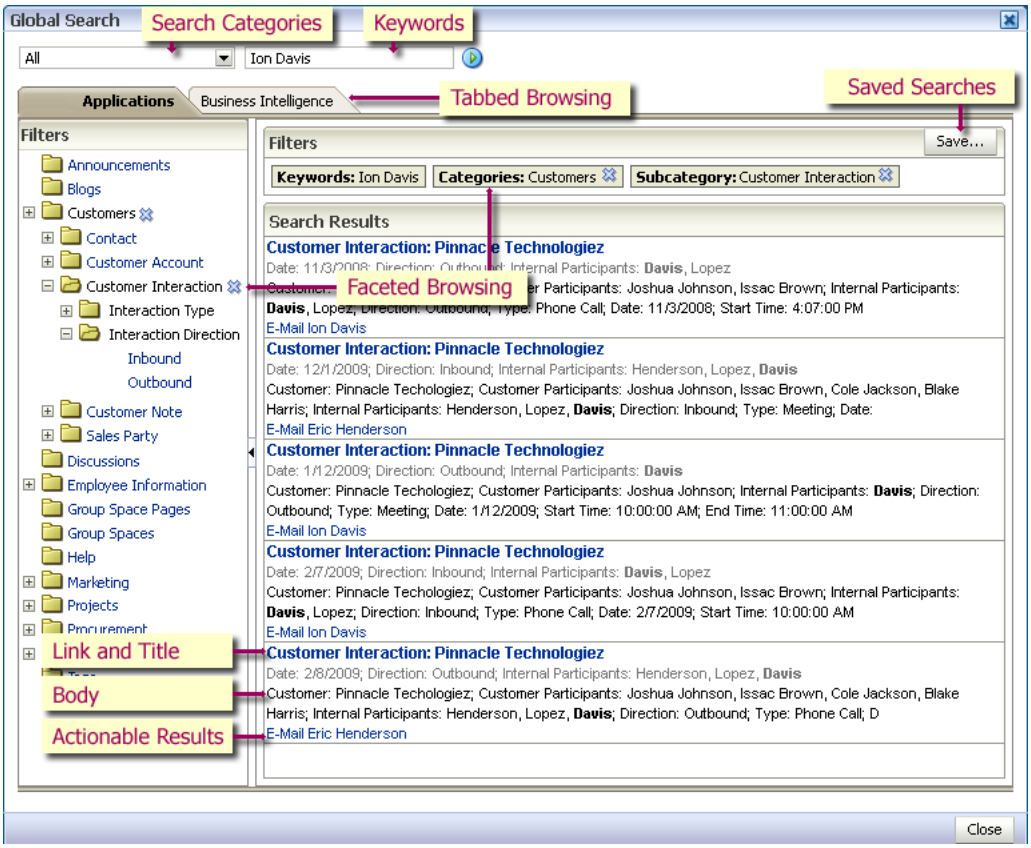
To configure search features, you can complete the following tasks:

- Define search result actions.
- Implement faceted navigation.

27.4.1 How to Define 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 Section 14.2.3, "How to Add Dynamic Main Area and Regional Area Task Flows to a Page". Figure 27–9 illustrates an example of the results of a filtered search.

Figure 27–9 Search Results with Action Links



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 27–3, 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 Section 14.15.6, "How to Use the Actionable Results API with Oracle Fusion Applications Search".

27.4.1.1 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.

27.4.1.2 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.

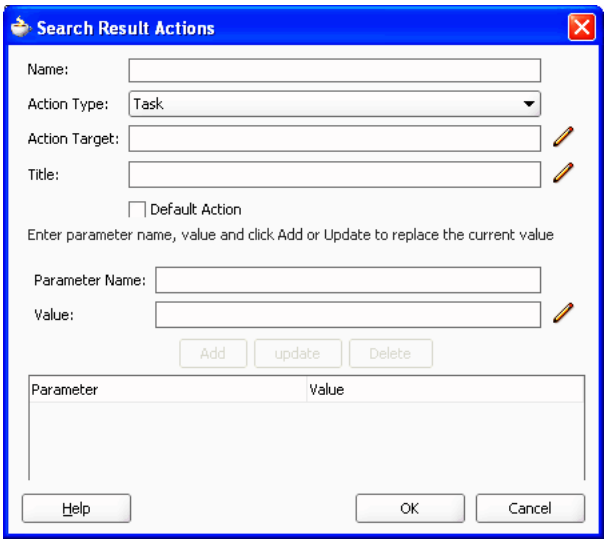
27.4.1.3 Adding Search Result Actions

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:

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 27–10.

Figure 27–10 Search Result Actions Dialog



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 Section 27.2.1, "How to Use 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 Section 27.4.1.4, "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 Section 27.2.1, "How to Use 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 Section 27.2.1, "How to Use 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.

27.4.1.4 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:

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.

27.4.1.5 Modifying 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:

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 27–10.

   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 Section 27.4.1.3, "Adding Search Result Actions".

   Note: You must recrawl the data if you reference a new attribute that is marked as stored.
6. Click **OK**.
7. Save the view object.

27.4.1.6 Deleting Search Result Actions

You can delete search result actions to remove action links that display with the search results.

To delete search result actions:

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.

27.4.2 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.

27.4.3 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 in order to save the action.

27.4.4 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 better 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.

For more information, see the "Working with List of Values (LOV) in View Object Attributes" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

27.4.4.1 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:

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.

27.4.4.2 Constraining View Objects by Stored Attributes

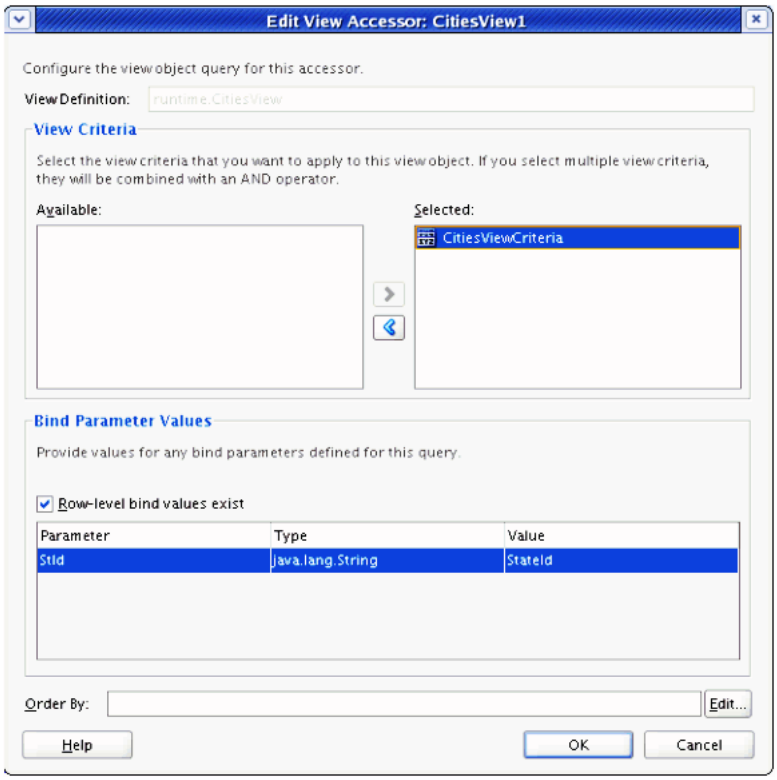
Constraining the `CitiesView` object by `StateID` makes sure that only the cities of the selected state are returned. Configure the `CitiesView1` view accessor on `EmpView` so that the view criteria on `CitiesView` is applied when the user queries using `CitiesView1`. Based on the view criteria, only 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:

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 27–11.

Figure 27–11 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.

27.4.4.3 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 27–3, 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.

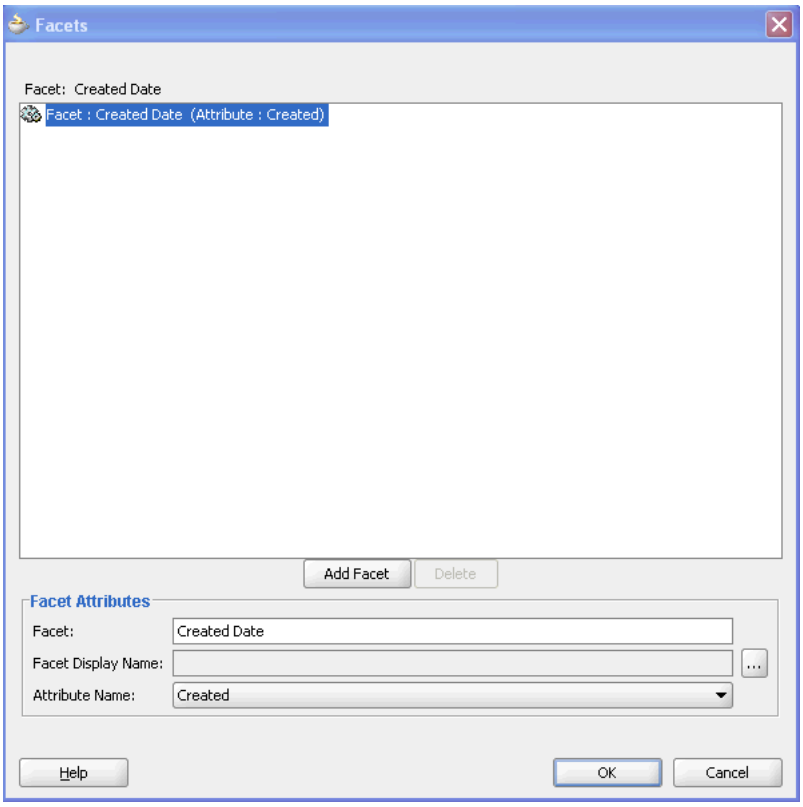
Before you begin:

Define LOVs for stored view object attributes. For information, see Section 27.4.4.1, "Defining Lists of Values".

To add search facets:

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 27–12.

Figure 27–12 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 Section 27.4.4.5, "Using the Select Text Resource Dialog to Select a Matching Text Resource" or Section 27.4.4.6, "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 Section 31.10, "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 Section 27.2.6.1, "Making 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.

27.4.4.4 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 27–7) 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.

To set up a View Object with support for child attributes in facets:

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 27–7.
7. Define the facet hierarchy.

27.4.4.5 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:

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.

27.4.4.6 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:

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**.

27.4.4.7 Modifying 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:

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 27–12.
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 Section 27.4.4.3, "Creating Search Facets".
   - Add child facets. For more information, see Section 27.4.4.3, "Creating Search Facets".
   - Delete child facets. For more information, see Section 27.4.4.9, "Deleting Child Search Facets".
7. Click **OK**.
8. Save the view object.

27.4.4.8 Deleting 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:

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.

27.4.4.9 Deleting Child Search Facets

You can delete child search facets to remove them from the facet tree.

To delete child search facets:

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 27–12.
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.

27.4.4.10 Defining Facets That Support Ranges

You can define facets to support ranges for numbers and dates.

To define facets that support ranges

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:

   ```
   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:

   ```
   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.

27.4.4.11 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

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:

```
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.

27.4.5 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.

27.4.6 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.

27.5 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.

27.5.1 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 30–1, can also be set as system parameters to apply the values to all searchable objects. For information, see Section 30.2.4, "How to Modify the Run Configuration of the View-Controller Project".

To configure customer properties for searchable objects:

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 30–1.

27.5.2 How to Make 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 27–7, in the *`view_object_name`*`.xml` file.

Example 27–7 Custom Property for Making Searchable Objects Public

```
<Properties>
    <CustomProperties>
        <Property
            Name="oracle.ecsf.searchableobject.public"
            Value="true"/>
    </CustomProperties>
</Properties>
```

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.

28

Configuring ECSF Security

This chapter describes how to configure security for ECSF.

This chapter includes the following sections:

28.1 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 in order to prevent denial-of-service (DoS) attacks, which makes the servlets unavailable. For information, see *Oracle Fusion Middleware Configuring Server Environments for Oracle WebLogic Server*.

28.2 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. For more information about CSF in Oracle WebLogic Server, see *Oracle Fusion Middleware Securing Oracle WebLogic Server*.

28.2.1 How to Add 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 28–1, is automatically added within the `<jazn-policy>` tag of the application's `jazn-data.xml` file located in `src/META-INF`.

Example 28–1 Permission Policy

```
<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>
```

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. For more information, see *Oracle Fusion Middleware Application Security Guide*.

28.2.2 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 WebLogic Server instance, then you must manually configure the application identities to integrate 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.

28.2.2.1 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 Section 30.2.4, "How to Modify the Run Configuration of the View-Controller Project".

28.2.2.2 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 *Oracle Fusion Applications Administrator's Guide*.

For example, set the user names for Oracle Fusion Customer Relationship Management to:

```
SES_ADMIN_USERNAME=eqsys

SES_QUERY_PROXY_USERNAME=FUSION_APPS_CRM_ECSF_SEARCH_APPID

ECSF_SECURITY_USERNAME=FUSION_APPS_CRM_SES_CRAWL_APPID
```

Once 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.

28.2.2.3 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 IdentityAssertion API, permissions must be granted to the three crawl users. The permission policies, shown in Example 28–2, are seeded in the `jazn-data.xml` file for Oracle Fusion applications and can be managed in Fusion Applications Control.

Example 28–2 Permission Policies for Application Identities

```
    <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>
            <permission>
                 <class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
                 <name>context=SYSTEM,mapName=oracle.apps.security,keyName=FUSION_APPS_HCM_SES_
CRAWL_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_FSCM_SES_
CRAWL_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_ECSF_SES_
ADMIN_APPID-KEY</name>
                <actions>*</actions>
            </permission>
            <permission>
                <class>oracle.security.jps.JpsPermission</class>
                <name>IdentityAssertion</name>
                <actions>execute</actions>
            </permission>
        </permissions>
    </grant>
```

The permissions allow ECSF to read and write credential store entries that are not part of the `oracle.ecsf` map.

28.3 Authorizing Users for Search Feeds

New grants are needed in order 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 28–3, inside the `<application>` section in the `<jazn-policy>` section.

Example 28–3 Grant for Search Feeds User Authorization

```
<permission>
      <class>oracle.adf.share.security.authorization.MethodPermission</class>
      <name>ECSF_All_Services</name>
      <actions>execute</actions>
</permission>
```

The grantee should be the users or roles that you want to authorize to use the search feeds, as shown in Example 28–4.

Example 28–4 Granting Permission to a Role

```
<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>
```

The example shows how `jazn-data.xml` is modified to grant the permission to a role.

28.4 Securing the Searchable Application Data

ECSF secures the searchable application data by authenticating and authorizing users who use ECSF to perform searches.

28.4.1 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:

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. For more information, see *Oracle Fusion Middleware Securing Oracle WebLogic Server*.
2. Create a separate user and add it to the Operators group in order to assign that user the Oracle WebLogic Server security role of Operator to obtain execute privileges on ECSF MBean operations. For more information, see *Oracle Fusion Middleware Securing Oracle WebLogic Server*.
3. Create an ECSF query proxy user. For more information, see the "Managing Search with Oracle Enterprise Crawl and Search Framework" chapter in the *Oracle Fusion Applications 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 *Oracle Fusion Applications Administrator's Guide*.

Note: ECSF also supports allowing authenticated users to search business objects with no security policies attached to them.

29

Validating and Testing Search Metadata

This chapter describes how to validate and test search metadata.

This chapter includes the following sections:

29.1 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.

29.2 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.

29.2.1 How to Validate Search Metadata

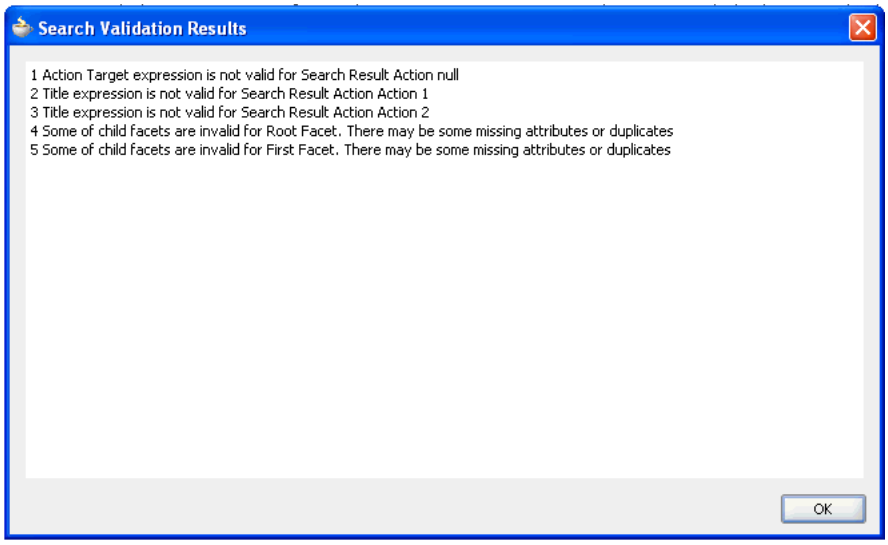
Use the Search navigation tab of the overview editor in JDeveloper, shown in Figure 27–3, to validate search metadata. If you identify any errors (changes to the view object metadata that are not reflected in the search metadata), you must manually fix them.

To validate search metadata through the Validate button:

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 29–1, appears.

Figure 29–1 Search Validations Dialog



   The dialog shows a list of validation errors.

3. Click **OK** when you finish viewing the errors.

29.3 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.

Before you begin:

1. Set the following Java option in the Run configuration dialog in order to test the searchable objects through a web browser: `-Doracle.ecsf.crawl.mode.debug=true`. For more information, see Section 30.2.4, "How to Modify the Run Configuration of the View-Controller Project".

2. Set up ECSF security. For information, see Chapter 28, "Configuring ECSF Security".
3. Create searchable objects. For information, see Chapter 27, "Creating Searchable Objects".
4. Run the ECSF feed servlet. For information, see Section 29.3.1, "How to Run the ECSF Feed Servlet".

29.3.1 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 29–1 when the ECSF Runtime Server library is added to the project.

Example 29–1 ECSF Feed Servlet, Feed Servlet Mapping, and Filter Mappings

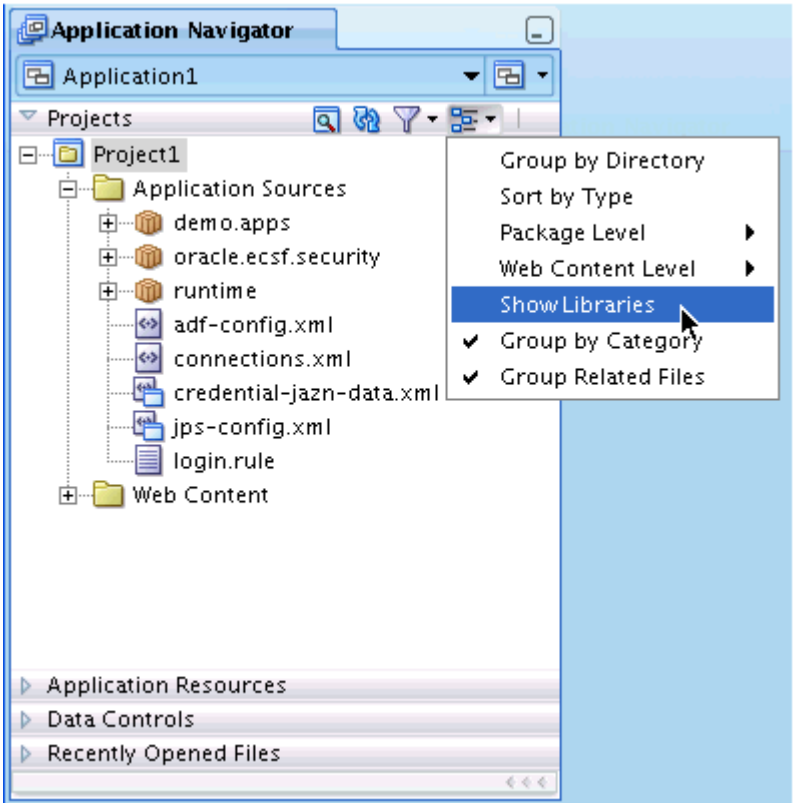
```
<servlet>
        <servlet-name>SearchFeedServlet</servlet-name>
        <servlet-class>oracle.ecsf.feed.SearchFeedServlet</servlet-class>
    </servlet>
<servlet-mapping>
        <servlet-name>SearchFeedServlet</servlet-name>
        <url-pattern>/searchfeedservlet/*</url-pattern>
    </servlet-mapping>
<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>
```

Run the ECSF feed servlet to make sure it runs properly.

To run the ECSF feed servlet:

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 29–2.

Figure 29–2 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`

   Note: The web page is an RSS feed. Depending on the browser you use, you may not be able to view the contents of the web page. If you cannot view the RSS feed, navigate to **View** > **Source** in the browser to view the feed.

   You can click the **Terminate** (red square) button to stop the ECSF feed servlet.

29.3.2 How to Test the Config Feed

To test the Config Feed, run the ECSF feed servlet with a modified URL.

To test the Config Feed:

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 `wlsserver.com`, and the fully qualified package path of `EmpVO` is `oracle.ecsf.EmpVO` (case sensitive), the resulting Config Feed URL would be `http://wlsserver.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 29–2.

Example 29–2 Sample Results of the Config Feed

```
- <rsscrawler xmlns="http://xmlns.example.com/search/rsscrawlerconfig">

<feedLocation>http://localhost:8988/approot/searchfeedservlet/runtime.EmpView/Cont
rolFeed</feedLocation>
  <feedType>controlFeed</feedType>
  <errorFeedLocation>/tmp</errorFeedLocation>
  <securityType>attributeBased</securityType>
  <securityAttribute name="DEPTNO" grant="true" />
  </rsscrawler>
```

   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 Section 27.2.3, "How to Make View Objects Searchable".

3. After adding the metadata, make sure to restart the Integrated WebLogic Server instance.

29.3.3 How to Test the Control Feed

To test the Control Feed, run the ECSF feed servlet with a modified URL.

To test the Control Feed:

1. Locate the URL provided in the `<feedLocation>` tag of the Config Feed. For information, see Section 29.3.2, "How to Test the Config Feed".

   In the Config Feed shown in Example 29–2, 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 29–3.

Example 29–3 Sample Results of the Control Feed

```
  <?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/ControlFee
d</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>
- <![CDATA[
http://example.com:8988/approot/searchfeedservlet/runtime.EmpView/DataFeed?tableNa
me=Emp&workUnitStart=AAHz%2BdADeAAA07UAAA&workUnitEnd=AAHz%2BdADeAAA07UAAV&type=RO
WID
  ]]>
  </link>
  </item>
  </channel>
  </rss>
```

29.3.4 How to Test the Data Feed

To test the Data Feed, run the ECSF feed servlet with a modified URL.

To test the Data Feed:

1. Locate the URL provided in the `<link>` tag under the `<item>` tag of the Control Feed. For information, see Section 29.3.3, "How to Test the Control Feed".

   In the Control Feed shown in Example 29–3, 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 29–4.

Example 29–4 Sample Results of the Data Feed

```
  <?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">
- <documentMetadata>
- <accessURL>
- <![CDATA[
http://example.com/EmpNo=7839
  ]]>
  </accessURL>
- <keywords>
- <![CDATA[
Employee department job salary data  KING
  ]]>
  </keywords>
- <summary>
- <![CDATA[
KING 1981-11-17T00:00:00.000Z
  ]]>
  </summary>
  <language>en</language>
- <docAttr name="ENAME">
- <![CDATA[
KING
  ]]>
  </docAttr>
- <docAttr name="CITY_ID">
- <![CDATA[
2
  ]]>
  </docAttr>
- <docAttr name="STATE_ID">
- <![CDATA[
1
  ]]>
  </docAttr>
  </documentMetadata>
- <documentAcl>
  <securityAttr name="DEPTNO">NO_SECURITY</securityAttr>
  </documentAcl>
- <documentInfo>
  <status>STATUS_OK_FOR_INDEX</status>
  </documentInfo>
- <documentContent>
- <content type="text/plain">
- <![CDATA[
Identification Number: 10.7839
  ]]>
  </content>
- <contentLink type="text/html">
- <![CDATA[
http://example.com:8988/approot/searchfeedservlet/runtime.EmpView/Attachment?schem
```

```
aName=null&tableName=runtime.EmpView&columnName=Attachment&keyCount=1&keyName0=EMP
NO&keyValue0=7839
  ]]>
  </contentLink>
  </documentContent>
  </itemDesc>
  </item>
```

29.3.5 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.

To reset the state of the feeds:

1. Recall the Config Feed URL of the searchable object. For information, see Section 29.3.2, "How to Test the Config Feed".
2. Append `?forceInitialCrawl=true` to the Config Feed URL, for example, `http://example.com:8988/approot/searchfeedservlet/runtime.EmpView/ConfigFeed?forceInitialCrawl=true`
3. Paste the resulting URL in your browser and refresh the web page.

30

Deploying and Crawling Searchable Objects

This chapter describes how to deploy searchable objects to the Oracle Enterprise Crawl and Search Framework (ECSF) application.

This chapter includes the following sections:

30.1 Introduction to Deploying and Crawling Searchable Objects

The ECSF application must include the searchable objects before you deploy it to the application server.

30.2 Deploying 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. In order 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`.

30.2.1 How to Deploy 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 30–1, is automatically added to the WebLogic deployment descriptor file (`weblogic-application.xml`).

Example 30–1 Reference to the ECSF Shared Library

```
<library-ref>
   <library-name>oracle.ecsf</library-name>
</library-ref>
```

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.

30.2.1.1 Updating 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:

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**.

30.2.1.2 Deploying 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 together with the shared library.

To deploy the ECSF shared library to the standalone WebLogic Server instance:

1. Extend the Oracle WebLogic Server domain by using the Oracle Fusion Middleware Configuration Wizard (execute `$WL_HOME/common/bin/config.sh`).

   For more information, see *Oracle Fusion Middleware Configuring Server Environments for Oracle WebLogic Server*.

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**

   For more information, see *Oracle Fusion Middleware Configuring Server Environments for Oracle WebLogic Server*.

4. When a new version of the ECSF shared library is available, you must redeploy it.

   a. Remove the old `oracle.ecsf` library.

      For information, see *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Online Help*.

   b. Install the library enterprise archive (EAR) file (`oracle/jdeveloper/ecsf/modules/oracle.ecsf_11.1.1/oracle.ecsf.ear`) with the name set as `oracle.ecsf`.

      For information, see *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Online Help*.

      Note: You can also redeploy the ECSF shared library by using the Oracle Fusion Middleware Configuration Wizard. For more information, see *Oracle Fusion Middleware Configuring Server Environments for Oracle WebLogic Server*.

30.2.2 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:

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**.

30.2.3 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:

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`.

30.2.4 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:

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 30–1 lists the ECSF system parameters. Separate each parameter with a space.

Note: The parameters in Table 30–1 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 Section 27.5, "Configuring Custom Properties for Searchable Objects".

Table 30–1 ECSF System Parameters

Parameter Name	Java Command-Line Entry	Description
`oracle.ecsf.admin.dataobject.synchinterval`	`-Doracle.ecsf.admin.dataobject.synchinterval=`*n*	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).
`oracle.ecsf.cache.expireseconds`	`-Doracle.ecsf.cache.expireseconds=`*n*	Defines the time (*n*), in seconds, for items to expire from the cache. The default value is `1800` (30 minutes).
`oracle.ecsf.connection.name`	`-Doracle.ecsf.connection.name=`*ConnectionName*	Specifies the name of the database connection to be used. If not specified, `SearchDB` is used.
`oracle.ecsf.context`	`-Doracle.ecsf.context=`*context*	Defines the ECSF context to be used. The value can be set to `oracle.ecsf.fusion.FusionSearchContextImpl` 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.
`oracle.ecsf.crawl.batch.size`	`-Doracle.ecsf.crawl.batch.size=`*n*	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.

Table 30–1 (Cont.) ECSF System Parameters

Parameter Name	Java Command-Line Entry	Description
`oracle.ecsf.crawl.datafeed.size`	`-Doracle.ecsf.crawl.datafeed.size=`*n*	Defines the size for the data feed. The value determines the number of documents (*n*) per data feed. The default size is 1000.
`oracle.ecsf.datasource.name`	`-Doracle.ecsf.datasource.name=`*DataSourceName*	Specifies the Java Database Connectivity (JDBC) data source name, such as `jdbc/SearchDBDS`.
`oracle.ecsf.datesplitter.mode`	`-Doracle.ecsf.datesplitter.mode=`*mode*	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.
`oracle.ecsf.maxfacetdefvalues`	`-Doracle.ecsf.maxfacetdefvalues=`*n*	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 a number of values (for example, USA, Canada, and so on) that exceeds the maximum. The default value is 1000.
`oracle.ecsf.max.links.depth`	`-Doracle.ecsf.max.links.depth=`*n*	Specifies the maximum number of view object hierarchy levels to limit the depth of search.
`oracle.ecsf.service.ws.timeout`	`-oracle.ecsf.service.ws.timeout=`*n*	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.

Table 30–1 (Cont.) ECSF System Parameters

Parameter Name	Java Command-Line Entry	Description

`oracle.ecsf.split.mode`	`-Doracle.ecsf.split.mode=db`
	or
	`-Doracle.ecsf.split.mode=simple`
	or
	`-Doracle.ecsf.split.mode=`
		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.

`oracle.ecsf.split.threshold`	`-Doracle.ecsf.split.threshold=`***n***	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.

6. Click **OK**.

   The run configuration is set to debug mode, and other ECSF system parameters are set.

30.2.5 How to Add 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/jdeveloper/soa/modules/oracle.soa.fabric_11.1.1/fabric-runtime.jar`
  - `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.

30.2.6 How to Deploy 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 Chapter 3, "Setting Up Your JDeveloper Workspace and Projects".

30.3 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.

30.3.1 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:

1. Deploy the index schedule.

   For information, see the "Deploy the Index Schedules" task in *Oracle Fusion Applications Administrator's Guide*.

2. Start the index schedule.

   For information, see the "Start the Index Schedules" task in *Oracle Fusion Applications 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.

   For information, see the *Oracle Secure Enterprise Search Administrator's Guide*.

31

Advanced Topics for ECSF

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:

31.1 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, localizing ECSF artifacts for international users, and information for troubleshooting ECSF.

31.2 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 Enterprise Content Management Suite repository.

31.2.1 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.applcore.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:

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 Section 18.2.2, "How to Create Attachment View Links".

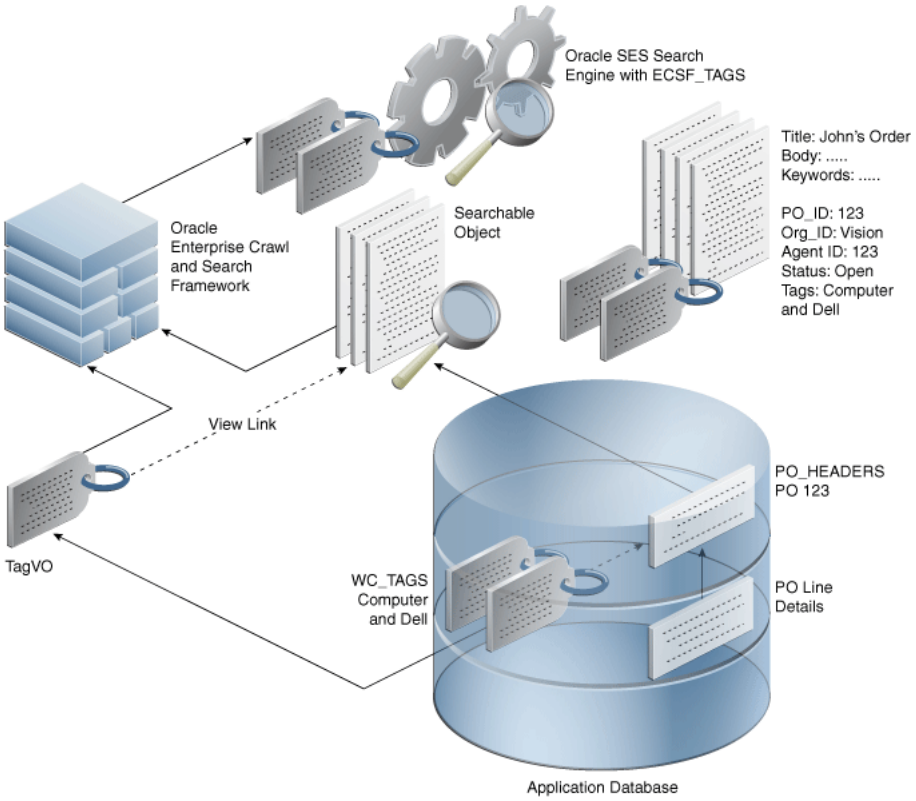
31.3 Enabling Search on WebCenter Tags

ECSF supports the capability to crawl searchable objects with Oracle WebCenter 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. For more information, see the "Integrating the Tags Service" chapter in *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

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 31–1 illustrates crawl time with tags.

Figure 31–1 Crawl Time With Tags

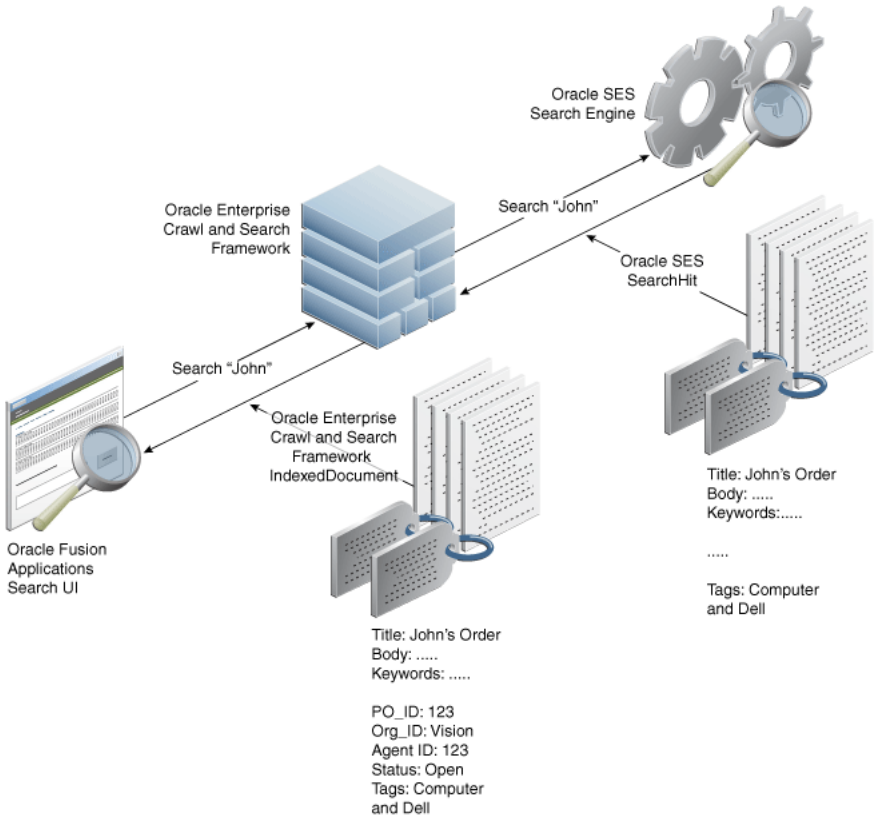


Purchase Order 123 has two tags, `Computer` and `Dell`. ECSF adds the tags for each record of the searchable object to the indexable document before the document is sent to Oracle SES for indexing. ECSF also creates a reserved attribute (of type string) called `ECSF_TAGS` to store tags in Oracle SES.

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 31–2 and Figure 31–3 illustrates the difference between query time without a tag and query time with a tag.

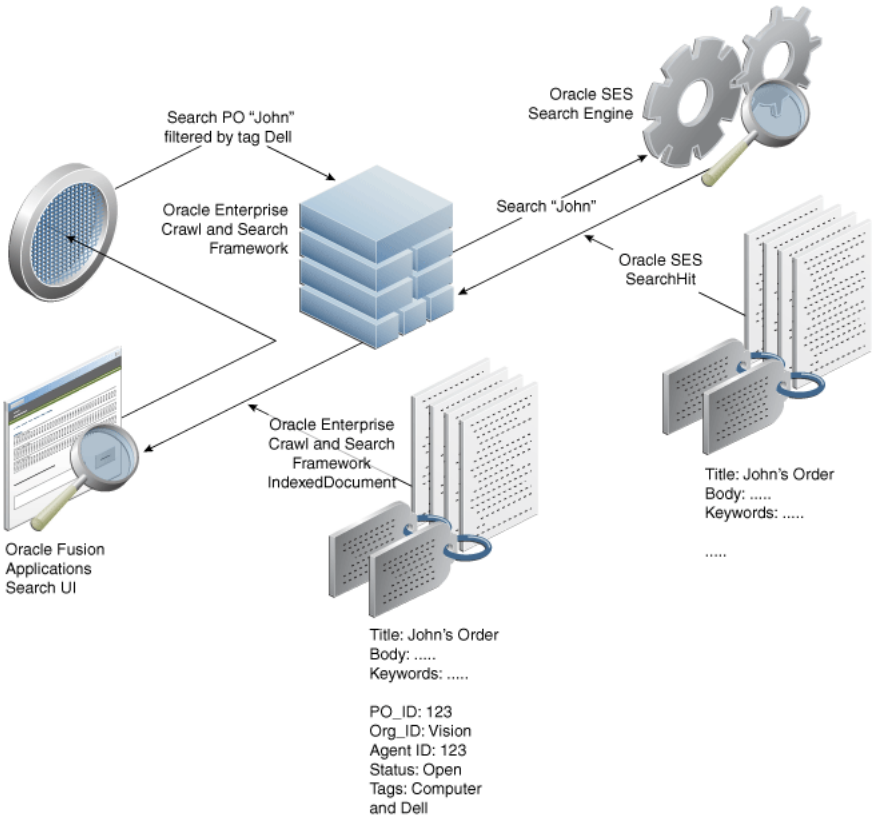
Note: Tags cannot be searched as individual entities.

Figure 31–2 Query Time Without Tag



In Figure 31–2, 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.

Figure 31–3 Query Time With Tag



In Figure 31–3, 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 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.

Perform the following tasks to enable search on WebCenter tags:

1. Create a view link between the searchable object and the `TagSVO`. For information, see Section 14.15.5, "How to Implement Tags in Oracle Fusion Applications 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.

You can also perform the following tasks to customize search on WebCenter tags:

1. Modify the tags in the indexable document.
2. Register change listeners.

31.3.1 How to Add Tags to Indexable Documents

In order 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 to indexable documents:

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.

You can implement code, such as the sample code shown in Example 31–1, in the search extension to extend `DefaultSearchPlugin`.

Example 31–1 Sample Code for Adding Tags to Indexable Documents

```
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
                 }
}
      }
   }
}
```

This extension adds three tags (`Black`, `White`, and `Stripes`) to a user named `Zebra`.

31.3.2 How to Add 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 31–2 illustrates how tags are used for querying.

Example 31–2 Sample API for Querying With Tags

```
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

//filter by tag that does not exist
        queryMetaData.clearTags();
        queryMetaData.addTag("Dots");
         searchHits = searchCtrl.runQuery(searchContext, queryMetaData);
//no result is found
   }
```

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.

31.3.3 How to Modify 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.

31.3.4 How to Register Change Listeners

You can extend the incremental crawling mechanism by registering change listeners, which use the WebCenter 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 31–3 illustrates a sample implementation of `ChangeListener`.

Example 31–3 Sample Implementation of ChangeListener

```
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)
       {
           setDone(true);
       }else
       {
//find how many has been changed since that date
            mCount = 100; //example
       }

   }
   protected List populate(SearchContext ctx)
```

```
   {
      List nextList = new ArrayList();
//the following code marks Zebra has been changed
PrimaryKey pk = new PrimaryKey();
Pk.put("ENAME", "Zebra");
NextList.add(pk);

setDone(true); //tells ECSF there are no more
      return nextList;
   }
}
```

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.

31.4 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. For more information about Oracle Business Intelligence Suite Enterprise Edition and its supported search functionality, see the "Managing Objects in the Oracle BI Presentation Catalog" chapter in *Oracle Business Intelligence Suite Enterprise Edition User's Guide*.

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.

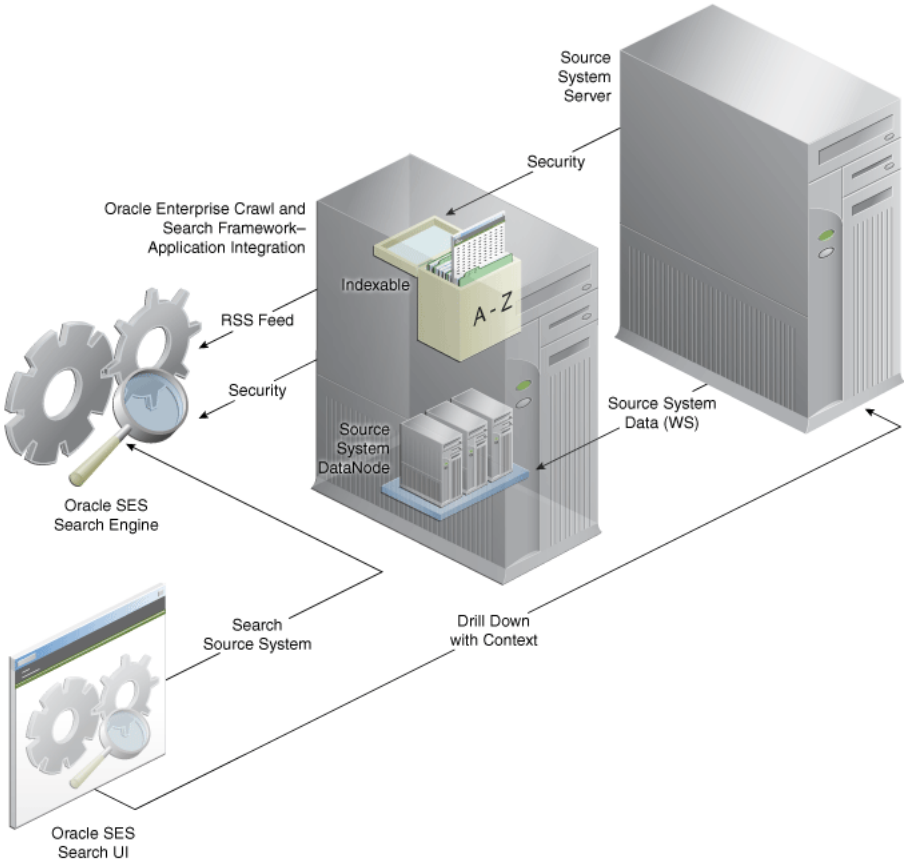
> **Note:** Crawling relies on the integrator's implementation that is based on their underlying data structures and accessibility of the items to be indexed.

> **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 31–4 illustrates the data flow for search on tree structure-based source systems.

Figure 31–4 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 Section 27.2.3, "How to Make 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 Section 31.2, "Enabling Search on Fusion File Attachments".

Enable search on tree structure-based source systems by:

1. Crawling tree structures

2. Integrating search functionality for tree structures
3. Implementing administration using ECSF interfaces

31.4.1 How to Crawl Tree Structures

ECSF offers Java classes that provide support for traversing tree structure-based data sources and identifying items for indexing. To implement search for these 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.

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 Chapter 55, "Initializing Oracle Fusion Application Data Using the Seed Data Loader".

31.4.1.1 Creating a Searchable Object

Create a searchable object by extending the `oracle.ecsf.data.tree.SearchableTreeObject` class, as shown in Example 31–4. 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.

Example 31–4 Sample Code for SearchableTreeDirectory Class

```
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";
    }

}
```

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 Section 31.4.1.3, "Extending AbstractTreeWalker".

31.4.1.2 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 31–5, for extracting document metadata. The tree node class models the tree structure repository.

Example 31–5 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 needs to 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);
   }
}
```

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.

31.4.1.3 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 31–6. `FileTreeCrawler` is invoked as a factory to create a crawlable tree node as defined in Section 31.4.1.2, "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.

Example 31–6 Sample Code for FileTreeCrawler Class

```
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();
    }
}
```

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 or not 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.

31.4.1.4 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.

> **Note:** ECSF uses the generic term ACL to describe how Oracle SES and ECSF pass security information and perform security checks by using the information described in the ACL.

ECSF is secured by a plugable 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 31–7 illustrates the skeleton of a security extension.

Example 31–7 Sample of Security Service

```
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[]{};
    }
}
```

The security extension is a Java class that implements a securable interface. There are five methods available. Example 31–8 illustrates the skeleton of such class.

Example 31–8 Sample Security Extension

```
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 agains 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[]{};
    }
}
```

Your search plug-in must be assigned to the searchable object, as shown in Example 31–4.

31.4.1.5 Implementing the Attachments Interface

Implementing the attachments interface, shown in Example 31–9, allows you to index binary files such as Word documents, Excel spreadsheets, PDF files, and so on.

Example 31–9 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 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;
    }
}
```

Once 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.

31.4.1.6 Deploying and Starting 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:

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 31–10.

Example 31–10 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>
```

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.

31.4.1.7 Configuring 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:

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`
   - **HTTP endpoint for authorization**: `http://`*`yourhost:port/appname/pathname`*`/SecurityService?engineInstID=`*`EngineInstanceID`*
   - **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.

d. For HTTP endpoint for authentication, enter `http://yourhost:port/appname/pathname/SecurityService`.

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 31–11, to query Oracle SES through API.

Example 31–11 Class Path with Oracle SES Client Java Archive

```
   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)
      {
         return null;
      }
      return searchHits;
   }
```

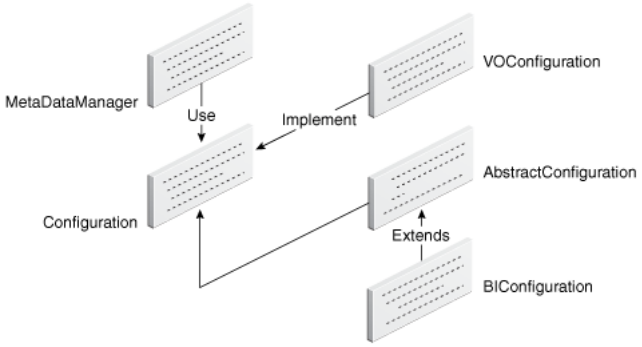
31.4.2 How to Integrate Search Functionality for Tree Structures

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 is capable of loading 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 31–5 illustrates the runtime architecture that includes the `Configuration` interface, which provides an alternative mechanism for loading runtime objects.

Figure 31–5 Runtime Architecture with Configuration Interface



A system property determines which configuration, `VOConfiguration` or a custom configuration, to use during runtime. In Figure 31–5, 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.

31.4.2.1 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,

```
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.

31.4.2.2 Using the Configuration Interface

The `Configuration` interface, shown in Example 31–12, contains the methods implemented by `VOConfiguration` or a custom configuration to load runtime objects.

Example 31–12 Configuration Interface

```
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();
}
```

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.

31.4.2.3 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 Section 31.4.2.5, "Extending AbstractConfiguration".

Example 31–13 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.

Example 31–13 Sample AbstractConfiguration Implementation

```
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;
         }
      }
      else
```

```
            {
               return null;
            }
         }
         catch (ClassNotFoundException e)
         {
            return null;
         }
      }

      public void reload()
      {
         //do nothing
      }

      private List<SearchableGroup> groups = null;
      private List<MetaEngineInstance> engines= new ArrayList<MetaEngineInstance>();
}
```

In this implementation, there is only one engine instance but following `MetaDataManager` convention, a list of engine instances is returned.

31.4.2.4 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 31–14 shows the sample implementation for `EmpView.java`.

Example 31–14 Sample Searchable Object Class

```
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 initalize document.
    * @param document
```

```
     */
   public final void setDocument(AbstractDocumentDefinition document)
   {
      super.setDocument(document);
      initDoc(document);
   }
   private void initDoc(AbstractDocumentDefinition documentDefintion)
   {
      FieldDefinitionImpl field = new FieldDefinitionImpl("Ename");
      field.setBinding("ENAME");
      field.setPrimaryKey(false);
      field.setStored(true);
      documentDefintion.addField(field);
      field = new FieldDefinitionImpl("Empno");
      field.setBinding("EMPNO");
      field.setPrimaryKey(true);
      field.setStored(true);
      documentDefintion.addField(field);
      field = new FieldDefinitionImpl(DocumentDefinition.ECSF_SO_NAME);
      field.setBinding(DocumentDefinition.ECSF_SO_NAME);
      field.setPrimaryKey(false);
      field.setStored(true);
      documentDefintion.addField(field);
   }
}
```

Each searchable object is loaded using a class of the same name.

31.4.2.5 Extending AbstractConfiguration

You can extend the `AbstractConfiguration` class with the `BIConfiguration` class, as shown in Example 31–15, to define your own way of loading runtime objects that is specific to your environment.

Example 31–15 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_SERVICE",

"http://sesserver.com:7777/search/query/OracleSearch");
      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";

}
```

31.5 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 or not 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.

- **ECSF_SVSEARCH Database Table**

  The existing ECSF_SVSearch database table will store the information for recent searches, using the SEARCH_TYPE column to store the Search Type.

31.5.1 How to Use the RecentSearchManager API

This class, shown in Example 31–16, manages the Recent Searches.

Example 31–16 RecentSearchManager API

```
/**
* 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,
                                        String callerCtx)
   throws SearchException

/**
* Deletes the specified Recent Search from the database
*
* @param ctx - the SearchContext
* @param recentSearch - the Recent Search to delete
* @throws SearchException
*/
 public void deleteRecentSearch(SearchContext ctx, SavedSearch savedSearch)
   throws SearchException

/**
* Deletes all of the recent searches for the current user specified in the
SearchContext
*
* @param ctx - the SearchContext
* @throws SearchException
*/
 public void deleteRecentSearchesForUser(SearchContext ctx)
   throws SearchException

/**
* Deletes all of the recent searches for the current user specified
* in the SearchContext with the specified callerCtx
*
* @param ctx the - SearchContext
* @param callerCtx - the caller context
* @throws SearchException
*/
 public void deleteRecentSearchesForUser(SearchContext ctx, String callerCtx)
   throws SearchException
```

31.5.2 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 RecentSsearch 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 or not 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:

```
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:

```
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:

```
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 RecentSeach and RecentSearchDetails dataobjects are used. A RecentSearch dataobject is just like SavedSearch except for the details method:

```
/**
 * Retrieve the RecentSearchDetails for this recent search.
 *
 * @return recent search details
 */
public RecentSearchDetails getRecentSearchDetails()
```

31.6 Setting 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 *Oracle Fusion Applications 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

31.6.1 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:

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.

31.6.2 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 for searchable objects:

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**.

31.6.3 How to Update 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 31–17.

Example 31–17 Reference to the ECSF Service Shared Library

```
<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>
```

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.

31.6.4 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 ECSF Runtime Library:

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.

31.6.5 How to Set 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.

31.6.5.1 Setting 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 using Java system properties:

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**.

31.6.5.2 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.

31.6.6 How to Package and Deploy 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.

31.6.6.1 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 postenterprise archive step that deletes all the files from the `deploy/lib` directory after the enterprise archive is packaged, resulting in a clean folder.

31.6.6.2 Running 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 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**.

31.6.7 How to Update 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 Section 31.6.6, "How to Package and Deploy 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 Section 31.6.6, "How to Package and Deploy the Search Application").

31.6.8 How to Set Up the ECSF Client Application for Federation

In order 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`

31.6.8.1 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`. For more information, see *Oracle Fusion Middleware Oracle WebLogic Scripting Tool*.

The following four new entries appear in `cwallet.sso` and `default-keystore.jks`:

```
> 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.

31.6.8.2 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`:

```
<serviceProvider type="KEY_STORE" name="keystore.provider"
        class="oracle.security.jps.internal.keystore.KeyStoreProvider">
</serviceProvider>
```

- Under `serviceInstances`:

```
<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`:

```
<serviceInstanceRef ref="keystore"/>
```

For more information, see *Oracle Fusion Middleware Oracle WebLogic Scripting Tool*.

31.6.8.3 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 31–18.

Example 31–18 Sample Proxy User Entry

```
createCred(map="oracle.wsm.security", key="test.user", user="weblogic",
password="weblogic1", desc="User credentials key")
```

The key (in this example, `test.user`) for the new entry is used in `connections.xml`.

31.6.8.4 Updating 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:

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 31–19.

Example 31–19 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>
               <wsconnection
description="http://localhost:7101/CrmSearchService/AppModuleSearchService?WSDL"
service="{/oracle/ecsf/service/query/common/}AppModuleSearchService">
                  <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">
                           <soap
addressUrl="http://localhost:7101/CrmSearchService/AppModuleSearchService"
xmlns="http://schemas.xmlsoap.org/wsdl/soap/"/>
                        </port>
                     </service>
                  </model>
               </wsconnection>
            </Contents>
```

```
         </XmlRefAddr>
      </RefAddresses>
   </Reference>
 <Reference name="{/oracle/ecsf/service/query/common/hcm/}SearchService"
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://localhost:7101/HcmSearchService/AppModuleSearchService?WSDL"
service="{/oracle/ecsf/service/query/common/}AppModuleSearchService">
                  <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">
                           <soap
addressUrl="http://localhost:7101/HcmSearchService/AppModuleSearchService"
xmlns="http://schemas.xmlsoap.org/wsdl/soap/"/>
                        </port>
                     </service>
                  </model>
               </wsconnection>
            </Contents>
         </XmlRefAddr>
      </RefAddresses>
   </Reference>
 <Reference name="{/oracle/ecsf/service/query/common/fscm/}SearchService"
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://localhost:7101/FscmSearchService/AppModuleSearchService?WSDL"
service="{/oracle/ecsf/service/query/common/}AppModuleSearchService">
                  <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">
                           <soap
addressUrl="http://localhost:7101/FscmSearchService/AppModuleSearchService"
xmlns="http://schemas.xmlsoap.org/wsdl/soap/"/>
                        </port>
                     </service>
                  </model>
               </wsconnection>
            </Contents>
         </XmlRefAddr>
      </RefAddresses>
 </Reference>
```

Web service security must be enforced by policy at the domain or instance level by configuration. For information, see *Oracle Fusion Middleware Oracle WebLogic Server Administration Console Online Help*.

3. Save.
4. Redeploy the client application.

31.6.9 How to Set the SearchContext Scope to GLOBAL

ECSF Query APIs can either invoke the ECSF service component internally (GLOBAL) or locally (LOCAL). In order 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 31–20.

Example 31–20 Client Methods

```
SearchContext ctx = ContextFactory.getSearchContext();
ctx.setScope("GLOBAL");
  ArrayList<SearchEngineInstance> engineInstances =
(ArrayList<SearchEngineInstance>)searchCtrl.getEngineInstances();
```

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 *Oracle Fusion Applications Administrator's Guide*.

31.6.10 How to Integrate Federation Across Oracle Fusion Applications Product Families

Use the ECSF API, as shown in Example 31–21, to integrate federation across Oracle Fusion Applications product families.

Example 31–21 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
      {
```

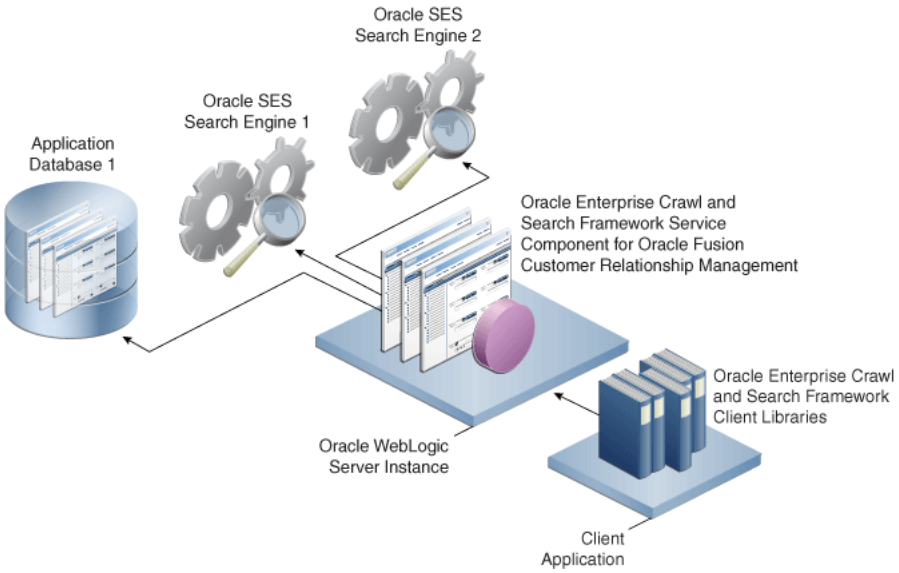
```
      searchHits = searchCtrl.runQuery(ctx, queryMetaData);
   }
   catch (Exception e)
   {
      e.printStackTrace();
   }
```

In the example, ECSF runtime gets the `runtime.EmpView` search category (search group) from an engine instance.

31.7 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 31–6, the ECSF client is connected to one ECSF service component, which depends on one database and two Oracle SES instances.

Figure 31–6 Federated Oracle SES



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 31–22 illustrates how you can integrate federation across Oracle SES instances.

Example 31–22 Sample API for Integrating Federated Oracle SES

```
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;
}
```

In the example, two Oracle SES search engine instances are defined.

31.8 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 31–23.

Example 31–23 Sample Code for Raising Events Using SQL

```
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);
```

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.

31.9 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.

31.9.1 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 31–1.

Table 31–1 ECSF Web Service Methods

Method	Description
`getSavedSearch(String userName, String savedSearchRequest)`	Returns a saved search based on the name passed in to the `savedSearchRequest` parameter.
`getSavedSearches(String userName, String savedSearchRequest)`	Returns the saved searches based on the caller context passed in to the `savedSearchRequest` parameter.
`saveSearch(String userName, String savedSearchRequest)`	Saves the search passed in to the `savedSearchRequest` parameter.
`deleteSearch(String userName, String savedSearchRequest)`	Deletes the saved search passed in to the `savedSearchRequest` parameter.
`getSavedSearchDetails(String userName, String savedSearchRequest)`	Returns the saved search details based on the saved search passed in to the `savedSearchRequest` parameter.
`search(String userName, String queryMetaDataRequest)`	Returns search hits based on the request passed in to the `QueryMetaData` parameter.
`getEngineInstances(String userName, String engineInstanceRequest)`	Returns the engine instances based on the engine type ID passed in to the `engineInstanceRequest` parameter.

31.9.2 ECSF Web Service WSDL and XSD

The Web Service Description Language (WSDL), shown in Example 31–24, defines the message endpoints and the request and reply messages of the ECSF web service. The XSD, shown in Example 31–25, 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 31–24 ECSF Web Service WSDL

```
<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: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:operation name="getSavedSearch">
            <wsdl:input message="tns:AppModuleSearchService_getSavedSearch"/>
            <wsdl:output message="tns:AppModuleSearchService_
getSavedSearchResponse"/>
            <wsdl:fault name="ServiceException"
message="errors:ServiceException"/>
        </wsdl:operation>
        <wsdl:operation name="getEngineInstances">
            <wsdl:input message="tns:AppModuleSearchService_getEngineInstances"/>
            <wsdl:output message="tns:AppModuleSearchService_
getEngineInstancesResponse"/>
            <wsdl:fault name="ServiceException"
message="errors:ServiceException"/>
        </wsdl:operation>
        <wsdl:operation name="getDisplayName">
            <wsdl:input message="tns:AppModuleSearchService_getDisplayName"/>
            <wsdl:output message="tns:AppModuleSearchService_
getDisplayNameResponse"/>
            <wsdl:fault name="ServiceException"
message="errors:ServiceException"/>
        </wsdl:operation>
        <wsdl:operation name="deleteSearch">
            <wsdl:input message="tns:AppModuleSearchService_deleteSearch"/>
            <wsdl:output message="tns:AppModuleSearchService_
deleteSearchResponse"/>
            <wsdl:fault name="ServiceException"
message="errors:ServiceException"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="AppModuleSearchServiceSoapHttp"
type="tns:AppModuleSearchService">
        <soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="search">
            <soap:operation
soapAction="/oracle/ecsf/service/query/common/search"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
```

```
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="ServiceException">
                <soap:fault name="ServiceException" use="literal"
encodingStyle=""/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="saveSearch">
            <soap:operation
soapAction="/oracle/ecsf/service/query/common/saveSearch"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="ServiceException">
                <soap:fault name="ServiceException" use="literal"
encodingStyle=""/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getSavedSearches">
            <soap:operation
soapAction="/oracle/ecsf/service/query/common/getSavedSearches"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="ServiceException">
                <soap:fault name="ServiceException" use="literal"
encodingStyle=""/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getSavedSearchDetails">
            <soap:operation
soapAction="/oracle/ecsf/service/query/common/getSavedSearchDetails"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="ServiceException">
                <soap:fault name="ServiceException" use="literal"
encodingStyle=""/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getSavedSearch">
            <soap:operation
soapAction="/oracle/ecsf/service/query/common/getSavedSearch"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="ServiceException">
```

```
                <soap:fault name="ServiceException" use="literal"
encodingStyle=""/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getEngineInstances">
            <soap:operation
soapAction="/oracle/ecsf/service/query/common/getEngineInstances"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="ServiceException">
                <soap:fault name="ServiceException" use="literal"
encodingStyle=""/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="getDisplayName">
            <soap:operation
soapAction="/oracle/ecsf/service/query/common/getDisplayName"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="ServiceException">
                <soap:fault name="ServiceException" use="literal"
encodingStyle=""/>
            </wsdl:fault>
        </wsdl:operation>
        <wsdl:operation name="deleteSearch">
            <soap:operation
soapAction="/oracle/ecsf/service/query/common/deleteSearch"/>
            <wsdl:input>
                <soap:body use="literal"/>
            </wsdl:input>
            <wsdl:output>
                <soap:body use="literal"/>
            </wsdl:output>
            <wsdl:fault name="ServiceException">
                <soap:fault name="ServiceException" use="literal"
encodingStyle=""/>
            </wsdl:fault>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="AppModuleSearchService">
        <wsdl:port name="AppModuleSearchServiceSoapHttpPort"
binding="tns:AppModuleSearchServiceSoapHttp">
            <soap:address
location="http://localhost:7101/Application1-ViewController-context-root/AppModule
SearchService"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Example 31–25 ECSF Web Service XSD

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
targetNamespace="/oracle/ecsf/service/query/common/types/"
     xmlns:tns="/oracle/ecsf/service/query/common/types/"
xmlns:ns0="http://xmlns.example.com/adf/svc/errors/">
    <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>
    <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>
<element name="getDisplayName">
    <complexType>
        <sequence>
            <element name="userName" type="string"/>
            <element name="displayNameRequest" type="string"/>
        </sequence>
    </complexType>
</element>
<element name="getDisplayNameResponse">
    <complexType>
        <sequence>
            <element name="result" type="string"/>
        </sequence>
    </complexType>
</element>
<element name="deleteSearch">
    <complexType>
        <sequence>
            <element name="userName" type="string"/>
            <element name="savedSearchRequest" type="string"/>
        </sequence>
    </complexType>
</element>
```

```
    <element name="deleteSearchResponse">
        <complexType>
            <sequence>
                <element name="result" type="string"/>
            </sequence>
        </complexType>
    </element>
</schema>
```

31.9.3 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

31.9.3.1 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 31–26.

Example 31–26 SavedSearch 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="savedSearch"/>
        <xsd:element name="name" type="xsd:string"/>
        <xsd:element name="callerctx" type="xsd:string"/>
        <xsd:element name="locale" type="xsd:string"/>
        <xsd:element name="scope" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="savedSearch">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="description" type="xsd:string"/>
        <xsd:element name="callerctx" type="xsd:string"/>
        <xsd:element name="query" type="xsd:string"/>
        <xsd:element name="ispublic" type="xsd:boolean"/>
        <xsd:element name="userid" type="xsd:string"/>
        <xsd:element name="detailsid" type="xsd:integer" minOccurs="0"/>
        <xsd:element ref="queryMetaData"/>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required"/>
```

```
      <xsd:attribute name="id" type="xsd:integer" use="required"/>
    </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: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>
```

The SavedSearch request XSD describes the set of rules that the request XMLs must follow in order to be valid. Examples of valid request XMLs include the following:

- `getSavedSearch()`

```
<request>
  <name>ECSF_JUNIT_SVSEARCH</name>
  <callerctx>null</callerctx>
  <locale>en_us</locale>
  <scope>LOCAL</scope> /*use GLOBAL for global scope*/
</request>
```

- `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>
      <![CDATA[]]>
    </callerctx>
    <query>
      <![CDATA[]]>
    </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>
  <name></name>
  <callerctx>null</callerctx>
  <locale>en_us</locale>
  <scope>LOCAL</scope> /*use GLOBAL for global scope*/
</request>
```

- deleteSearch()

```
<request>
  <savedSearch name="DeleteSavedSearch" id="100010033142848" compid="2">
    <description>
      <![CDATA[Junit Saved Search]]>
    </description>
    <callerctx>
      <![CDATA[CRM]]>
    </callerctx>
    <query>
      <![CDATA[%]]>
    </query>
    <ispublic>
      <![CDATA[false]]>
    </ispublic>
    <userid>
      <![CDATA[junit]]>
    </userid>
    <detailsid>
      <![CDATA[100010033142849]]>
    </detailsid>
    <queryMetaData>
      <query>
        <![CDATA[%]]>
      </query>
      <page>1</page>
      <lang>en</lang>
      <pageSize>10</pageSize>
      <categories>
        <category name="runtime.EmpView" eid="1" compid="3"></category>
      </categories>
    </queryMetaData>
  </savedSearch>
  <name></name>
  <callerctx>null</callerctx>
  <locale>en_us</locale>
  <scope>LOCAL</scope> /*use GLOBAL for global scope*/
</request>
```

- getSavedSearchDetails()

```
<request>
  <savedSearch name="SavedSearch" id="100010033142848" compid="2">
    <description>
      <![CDATA[Junit Federation Saved Search]]>
    </description>
    <callerctx>
      <![CDATA[CRM]]>
    </callerctx>
```

```
        <query>
          <![CDATA[%]]>
        </query>
        <ispublic>
          <![CDATA[false]]>
        </ispublic>
        <userid>
          <![CDATA[junit]]>
        </userid>
        <detailsid>
          <![CDATA[100010033142849]]>
        </detailsid>
      </savedSearch>
      <callerctx>null</callerctx>
      <locale>en_us</locale>
      <scope>LOCAL</scope> /*use GLOBAL for global scope*/
    </request>
```

31.9.3.2 QueryMetaData Request XSD

The request XML for the `search(String userName, String queryMetaDataRequest)` method is based on the QueryMetaData request XSD, shown in Example 31–27.

Example 31–27 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: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>
```

The QueryMetaData request XSD describes the set of rules that the request XMLs must follow in order to be valid. An example of a valid request XML for `search()` is:

```
<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>
  </queryMetaData>
  <locale>en_us</locale>
  <scope>LOCAL</scope> /*use GLOBAL for global scope*/
</request>
```

31.9.3.3 engineInstanceRequest Request XSD

The request XML for the `getEngineInstances(String userName, String engineInstanceRequest)` method is based on the engineInstanceRequest request XSD, shown in Example 31–28.

Example 31–28 engineInstanceRequest 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 name="enginetypeid"/>
        <xsd:element name="locale" type="xsd:string"/>
        <xsd:element name="scope" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

The engineInstanceRequest request XSD describes the set of rules that the request XMLs must follow in order to be valid. An example of a valid request XML for `getEngineInstances()` is:

```
<request>
  <enginetypeid>-1</enginetypeid>
  <locale>en_us</locale>
  <scope>LOCAL</scope> /*use GLOBAL for global scope*/
</request>
```

31.9.4 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 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>
```

31.9.4.1 getSavedSearch()

The web service response message for the method `String getSavedSearch(String userName, String savedSearchRequest)` is based on the following XSD:

```
<?xml 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:element name="description" type="xsd:string"/>
        <xsd:element name="callerctx" type="xsd:string"/>
        <xsd:element name="query" type="xsd:string"/>
        <xsd:element name="ispublic" type="xsd:boolean"/>
        <xsd:element name="userid" type="xsd:string"/>
        <xsd:element name="detailsid" type="xsd:integer" minOccurs="0"/>
        <xsd:element ref="queryMetaData"/>
      </xsd:sequence>
      <xsd:attribute name="name" type="xsd:string" use="required"/>
      <xsd:attribute name="id" type="xsd:integer" use="required"/>
    </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: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>
```

31.9.4.2 getSavedSearches()

The web service response message for the method `String getSavedSearches(String userName, String savedSearchRequest)` is based on the following XSD:

```
<?xml 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>
```

31.9.4.3 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 Section 31.9.4.1, "getSavedSearch()".

31.9.4.4 deleteSearch()

The web service response message for the method `String deleteSearch(String userName, String savedSearchRequest)` is the string `success`.

31.9.4.5 getSavedSearchDetails

The web service response message for the method `String getSavedSearchDetails(String userName, String savedSearchRequest)` is based on the following XSD:

```
<?xml 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="savedSearchDetails">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="queryMetaData"/>
      </xsd:sequence>
      <xsd:attribute name="savedSearchId" type="xsd:integer" use="required"/>
      <xsd:attribute name="id" type="xsd:integer" use="required"/>
    </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: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>
```

31.9.4.6 search()

The web service response message for the method `String search(String userName, String queryMetaDataRequest)` is based on the following XSD:

```
<?xml 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 name="hits" 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="searchableObjects" minOccurs="0">
                            <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:attribute name="displayName"
type="xsd:string" use="required"/>
                                  </xsd:complexType>
                                </xsd:element>
                              </xsd:sequence>
                            </xsd:complexType>
                          </xsd:element>
                        </xsd:sequence>
                        <xsd:attribute name="name" type="xsd:string"
use="required"/>
                        <xsd:attribute name="id" type="xsd:integer"
use="required"/>
                        <xsd:attribute name="eid" type="xsd:integer"
use="required"/>
                        <xsd:attribute name="external" type="xsd:string"
use="required"/>
                        <xsd:attribute name="displayName" type="xsd:string"
use="required"/>
                        <xsd:attribute name="scope" type="xsd:string"
use="required"/>
                        <xsd:attribute name="applid" type="xsd:string"
```

```
use="required"/>
                      </xsd:complexType>
                    </xsd:element>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
              <xsd:element name="altwords" type="xsd:string" minOccurs="0"/>
              <xsd:element name="timespent" type="xsd:integer"/>
              <xsd:element name="facets" minOccurs="0">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="facet" maxOccurs="unbounded" minOccurs="0">
                      <xsd:complexType>
                        <xsd:sequence>
                          <xsd:element name="pathEntry" maxOccurs="unbounded"
minOccurs="0">
                            <xsd:complexType>
                              <xsd:attribute name="pname" type="xsd:string"
use="required"/>
                              <xsd:attribute name="pdispname" type="xsd:string"
use="required"/>
                              <xsd:attribute name="pisleaf" type="xsd:boolean"
use="required"/>
                              <xsd:attribute name="value" type="xsd:string"
use="required"/>
                              <xsd:attribute name="count" type="xsd:integer"
use="required"/>
                              <xsd:attribute name="displayValue" type="xsd:string"
use="required"/>
                            </xsd:complexType>
                          </xsd:element>
                          <xsd:element name="entries" minOccurs="0">
                            <xsd:complexType>
                              <xsd:sequence>
                                <xsd:element name="entry" maxOccurs="unbounded">
                                  <xsd:complexType>
                                    <xsd:attribute name="value" type="xsd:string"
use="required"/>
                                    <xsd:attribute name="displayValue"
type="xsd:string" use="required"/>
                                  </xsd:complexType>
                                </xsd:element>
                              </xsd:sequence>
                            </xsd:complexType>
                          </xsd:element>
                        </xsd:sequence>
                        <xsd:attribute name="root" type="xsd:string"
use="required"/>
                        <xsd:attribute name="name" type="xsd:string"
use="required"/>
                        <xsd:attribute name="dispname" type="xsd:string"
use="required"/>
                        <xsd:attribute name="isleaf" type="xsd:boolean"
use="required"/>
                      </xsd:complexType>
                    </xsd:element>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
              <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:element name="results" minOccurs="0">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="result" maxOccurs="unbounded">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="title" type="xsd:string"/>
                    <xsd:element name="displayUrl" type="xsd:string"/>
                    <xsd:element name="score" type="xsd:integer"/>
                    <xsd:element name="searchableObject" minOccurs="0">
                      <xsd:complexType>
                        <xsd:attribute name="name" type="xsd:string"
use="required"/>
                        <xsd:attribute name="eid" type="xsd:integer"
use="required"/>
                      </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="content" type="xsd:string"/>
                    <xsd:element name="keywords" type="xsd:string" minOccurs="0"/>
                    <xsd:element name="lang" type="xsd:string"/>
                    <xsd:element name="attributes">
                      <xsd:complexType>
                        <xsd:sequence>
                          <xsd:element name="attribute" maxOccurs="unbounded"
minOccurs="0">
                            <xsd:complexType>
                              <xsd:simpleContent>
                                <xsd:extension base="xsd:string">
                                  <xsd:attribute name="name" type="xsd:string"
use="required"/>
                                  <xsd:attribute name="binding" type="xsd:string"
use="required"/>
                                  <xsd:attribute name="type" type="xsd:string"
use="required"/>
                                  <xsd:attribute name="displayName"
type="xsd:string" use="required"/>
                                </xsd:extension>
                              </xsd:simpleContent>
```

```
                            </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:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

31.9.4.7 getEngineInstances()

The web service response message for the method `String getEngineInstances(String userName, String engineInstanceRequest)` is based on the following 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="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: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:element name="facetDefs" minOccurs="0">
                                <xsd:complexType>
                                  <xsd:sequence>
                                    <xsd:element name="facetDef"
maxOccurs="unbounded">
                                      <xsd:complexType>
                                        <xsd:sequence>
                                          <xsd:element name="facetEntryDef"
maxOccurs="unbounded" minOccurs="0">
                                            <xsd:complexType>
                                              <xsd:attribute name="value"
type="xsd:string" use="required"/>
                                              <xsd:attribute name="displayValue"
type="xsd:string" use="required"/>
                                            </xsd:complexType>
                                          </xsd:element>
                                        </xsd:sequence>
                                        <xsd:attribute name="name"
type="xsd:string" use="required"/>
                                        <xsd:attribute name="binding"
type="xsd:string" use="required"/>
                                        <xsd:attribute name="displayName"
type="xsd:string" use="required"/>
                                        <xsd:attribute name="isleaf"
type="xsd:boolean" use="required"/>
                                      </xsd:complexType>
                                    </xsd:element>
                                  </xsd:sequence>
                                </xsd:complexType>
                              </xsd:element>
                              <xsd:element name="actionDefs" minOccurs="0">
                                <xsd:complexType>
                                  <xsd:sequence>
                                    <xsd:element name="actionDef"
maxOccurs="unbounded" minOccurs="0">
                                      <xsd:complexType>
                                        <xsd:sequence>
                                          <xsd:element name="target"
type="xsd:string"/>
                                          <xsd:element name="title"
type="xsd:string"/>
                                          <xsd:element name="params"
minOccurs="0">
                                            <xsd:complexType>
                                              <xsd:sequence>
                                                <xsd:element name="param"
maxOccurs="unbounded" minOccurs="0">
                                                  <xsd:complexType>
                                                    <xsd:simpleContent>
                                                      <xsd:extension
base="xsd:string">
                                                        <xsd:attribute
name="name" type="xsd:string" use="required"/>
                                                      </xsd:extension>
                                                    </xsd:simpleContent>
                                                  </xsd:complexType>
```

```
                                                    </xsd:element>
                                                  </xsd:sequence>
                                                </xsd:complexType>
                                              </xsd:element>
                                            </xsd:sequence>
                                            <xsd:attribute name="name"
type="xsd:string" use="required"/>
                                            <xsd:attribute name="isDefault"
type="xsd:string" use="required"/>
                                            <xsd:attribute name="type"
type="xsd:string" use="required"/>
                                          </xsd:complexType>
                                        </xsd:element>
                                      </xsd:sequence>
                                    </xsd:complexType>
                                  </xsd:element>
                                </xsd:sequence>
                                <xsd:attribute name="name" type="xsd:string"
use="required"/>
                                <xsd:attribute name="id" type="xsd:integer"
use="required"/>
                                <xsd:attribute name="displayName" type="xsd:string"
use="required"/>
                              </xsd:complexType>
                            </xsd:element>
                          </xsd:sequence>
                        </xsd:complexType>
                      </xsd:element>
                    </xsd:sequence>
                    <xsd:attribute name="name" type="xsd:string" use="required"/>
                    <xsd:attribute name="eid" type="xsd:integer" use="required"/>
                    <xsd:attribute name="id" type="xsd:integer" use="required"/>
                    <xsd:attribute name="external" type="xsd:boolean"
use="required"/>
                    <xsd:attribute name="displayName" type="xsd:string"/>
                    <xsd:attribute name="scope" type="xsd:string" use="required"/>
                    <xsd:attribute name="applid" type="xsd:string" use="required"/>
                  </xsd:complexType>
                </xsd:element>
              </xsd:sequence>
              <xsd:attribute name="name" type="xsd:string" use="required"/>
              <xsd:attribute name="id" type="xsd:integer" use="required"/>
              <xsd:attribute name="engineType" type="xsd:string" use="required"/>
              <xsd:attribute name="engineTypeId" type="xsd:integer" use="required"/>
              <xsd:attribute name="displayName" type="xsd:string" use="required"/>
            </xsd:complexType>
          </xsd:element>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

31.9.5 How to Invoke 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.

31.9.5.1 Creating 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`
   - `default-keystore.jks`

31.9.5.2 Modifying the AppModuleSearchServiceSoapHttpPortClient Class

After you create the JAX-WS proxy, modify the `AppModuleSearchServiceSoapHttpPortClient` class to include the calls to web service methods. Example 31–29 shows an example of a modified class.

Example 31–29 Sample Client Class

```
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.
USERNAME_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>LO
CAL</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-U
S</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);
      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 exeception : " + 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);
      String fullname = u.getFile();
      return fullname;
   }
}
```

The sample client class illustrates how each of the methods are called and how the XMLs are deserialized.

31.10 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 in order to display the messages in the appropriate language for the user.

31.10.1 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 param 2,...)` 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.

31.10.1.1 Associating Resource Bundles to View Objects

In order 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 to view objects:

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.

31.10.1.2 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.

31.10.1.3 Associating 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:

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.

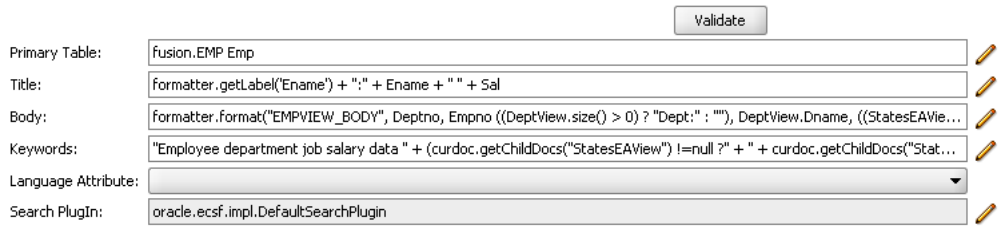
6. In the **Display Value** field, enter a string to associate with the key in the page's resource bundle.
7. In the **Key** field, enter a string to uniquely identify a locale-specific object in the resource bundle.
8. In the **Description** field, enter a description of any length for the key and value 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.

31.10.1.4 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 31–7.

Figure 31–7 Example of Calling the getLabel() Function



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.

31.10.2 How to Localize 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.

31.10.2.1 Configuring LOVs for Localization Using the VL Table

Facets are displayed in a tree.

`Facet Display Name`

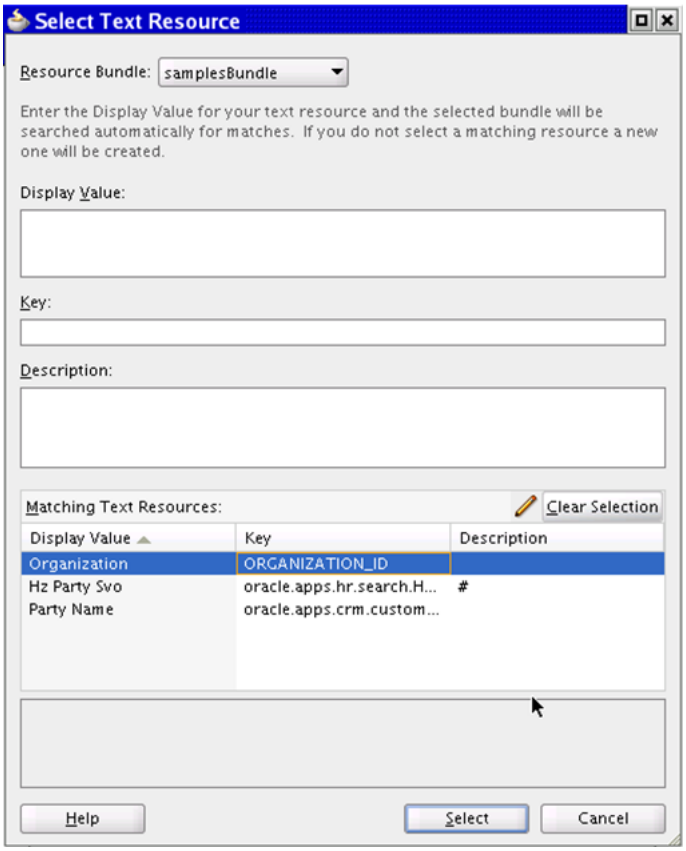
```
        Facet Entry Display Name
        Facet Entry Display Name
        ...
```

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.

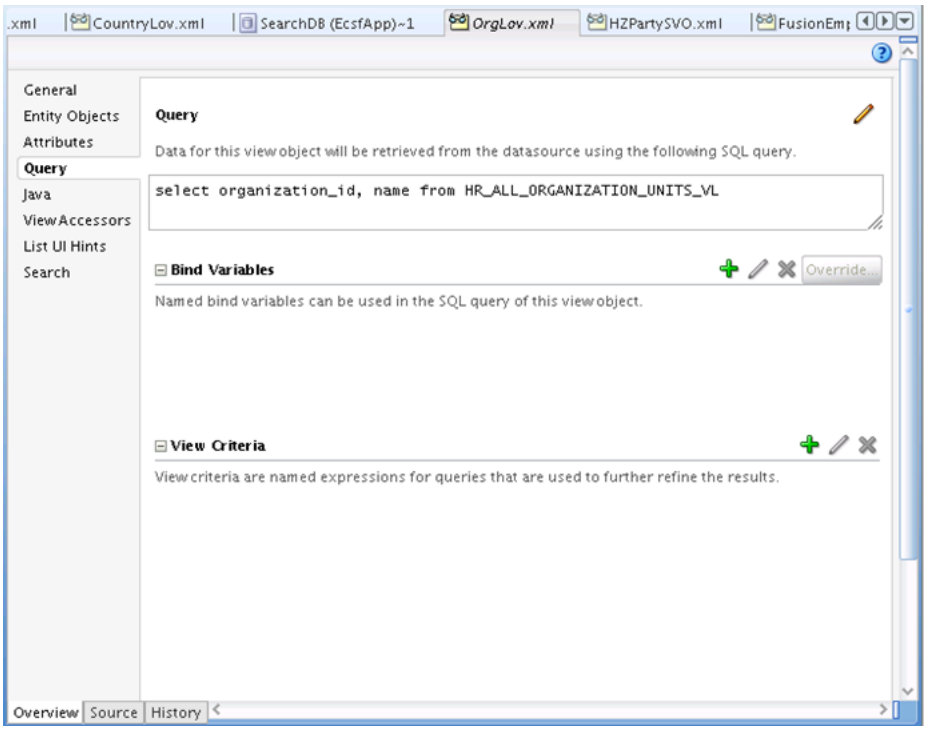
For example:

```
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 Section 27.4.4, "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`.

31.10.2.2 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.

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 Section 27.4.4, "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 LOV display values:

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`). For information, see `http://ftp.ics.uci.edu/pub/ietf/http/related/iso639.txt`.
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:

   ```
   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.

31.10.3 How to Localize 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 Chapter 55, "Initializing Oracle Fusion Application Data Using the Seed Data Loader".

31.10.4 How to Localize 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 Section 31.10.6, "How to Determine Locale".

31.10.5 How to Localize 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 in order 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 31–30 illustrates the use of a Groovy function with Java Text Formatting to form the body for localization.

Example 31–30 Using Groovy Function to Translate the Body

```
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);
}
```

`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 31–2 for a sample view object, Table 31–3 for a sample child view object, and Example 31–31 for a sample resource bundle.

Table 31–2 Sample Parent View Object (Employee)

ID	FirstName	LastName	EmailAddress	ZipCode
1	John	Wayne	john.wayne@example.com	94065
2	Chris	Smith	chris.smith@example.com	94066

Table 31–3 Sample Child View Object (Reports)

ParentID	FIRST_NAME	LAST_NAME
1	John	Doe
1	Jake	Ray
2	Bob	Barley
2	Chris	Carpenter

Example 31–31 Sample Resource Bundle

```
id=EMPLOYEE_MESSAGE,message="Employee {0} {1} with email address {2} works in
zipcode {4}"
id=REPORTS_TEMPLATE,message=" direct reports {0} {1}"
```

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.

31.10.6 How to Determine Locale

To display messages in the appropriate language for the user, ECSF must determine the locale. How ECSF determines locale varies by module.

31.10.6.1 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.

31.10.6.2 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()`.

31.10.6.3 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 Section 27.2.3.1, "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.

31.10.6.4 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 31–32, is used.

Example 31–32 Default Locale for Searcher

```
ADFContext adfContext = ADFContext.getCurrent();
Locale mLocale = (adfContext != null && adfContext.getLocale() != null) ?
adfContext.getLocale() : Locale.getDefault();
```

31.11 Troubleshooting ECSF

This section describes common problems that you might encounter when using ECSF and explains how to solve them.

31.11.1 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 Oracle WebLogic Server or Running the Application
- Blank Oracle ADF/UI Shell Pages
- Memory Leak on ThreadLocal Variable (SearchContext)

31.11.1.1 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.

31.11.1.2 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 *Oracle Fusion Applications Administrator's Guide*.
- Through the browser, append `?forceInitialCrawl=true` to the config feed.

31.11.1.3 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.

31.11.1.4 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.

31.11.1.5 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`.

31.11.1.6 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.applcore.globalSearch.ui.ecsf.ECSFSearchUtils.getSearchControl(ECS
FSearchUtils.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.

31.11.1.7 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.

31.11.1.8 How to Check 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:

```
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:

```
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:

```
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.

31.11.1.9 How to Crawl 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 needs to 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.

31.11.1.10 "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.

31.11.1.11 "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
```

31.11.1.12 "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:

```
at
oracle.ecsf.service.query.ws.marshaller.ServiceUtil.toError(ServiceUtil.java:1499)
```

31.11.1.13 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.

31.11.1.14 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.

31.11.1.15 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/rfd**. 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.

31.11.1.16 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.

31.11.1.17 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.

- Connect to WLST.
- Run the command:

  ```
  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
```

31.11.2 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. For information, see the "Configuring Settings for Log Files" chapter in the *Oracle Fusion Middleware Administrator's Guide*.

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`

31.11.3 Need More Help?

For additional assistance, you can send your inquiries to `HELPECSF_US@oracle.com`.

Part VI

Common Service Use Cases and Design Patterns

This part of the developer's guide describes the fundamental patterns that Oracle Fusion application developers should use when building applications involving Oracle Application Development Framework (Oracle ADF) and the Oracle SOA platform. The majority of these use cases fall into three basic patterns:

- The use of business events to initiate business processes
- Orchestrating over business logic implemented with Oracle ADF, Java, PL/SQL, and SOA composite applications
- Modeling human task flows in ADF applications

In addition to these three core categories, other chapters within this part provide guidance on a few less common patterns that might be useful to applications developers.

Each chapter in this section describes a use case and its associated recommended design pattern, along with procedures for implementing the design pattern, recommended validation procedures, and troubleshooting tips.

> **Note:** When carrying out the procedures described in the following chapters, use the Default/All technologies role for any SOA-related activity.

This part contains the following chapters:

32

Initiating 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).

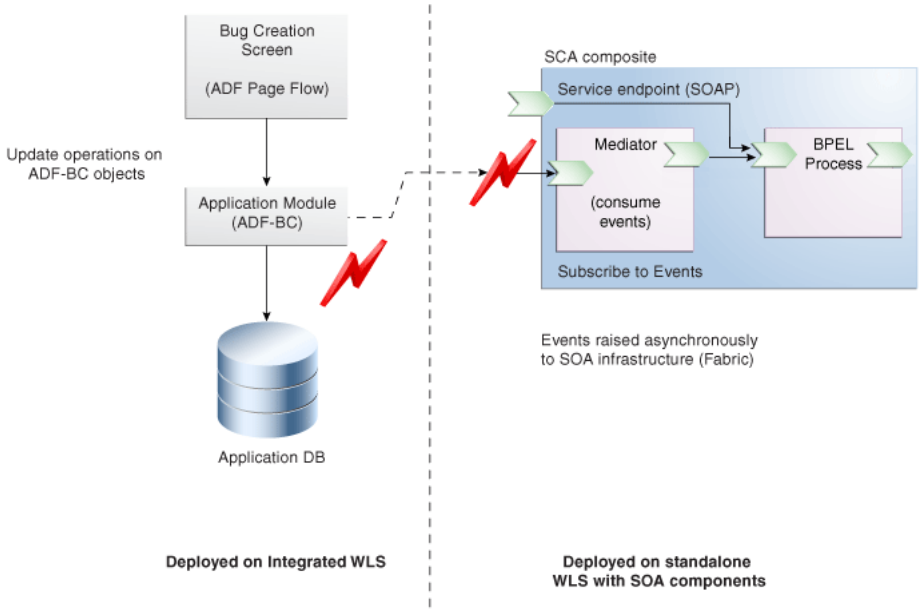
32.1 Introduction to the Recommended Design Pattern

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). For more information about the EDN, see the chapter "Using Business Events and the Event Delivery Network" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

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 32–1 illustrates how these work together.

Figure 32–1 Mediator Service Component Subscribes to an Event and Invokes BPEL Service Component



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. 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 *Oracle Fusion Middleware User's Guide for Technology Adapters*.

32.2 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. These development approaches should be used 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.**

32.3 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.

32.4 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.

   For more information on creating events on entity objects, see the chapter "Creating a Business Domain Layer Using Entity Objects" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. For more information about using business events, see the chapter "Using Business Events and the Event Delivery Network" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

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.

   For more information about creating the view, see "Part IV: Creating a Databound Web User Interface" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

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 32–1to the `WEB-INF/web.xml` file.

Example 32–1 Editing the web.xml File

```
<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>
```

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 32–2.

Example 32–2 Mapping Global JMS Resources to Local JNDI Names

```
<?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>
```

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` in order to include the event publishing APIs bundled in the relevant JAR files.

   Add the code shown in Example 32–3 to the `weblogic-application.xml` file.

Example 32–3 Add a Reference to oracle.soa.workflow.wc to the weblogic-application.xml File

```
<library-ref>
    <library-name>
      oracle.soa.workflow.wc
    </library-name>
  </library-ref>
```

5. In Oracle JDeveloper, create a SOA composite application project and add a mediator service component.

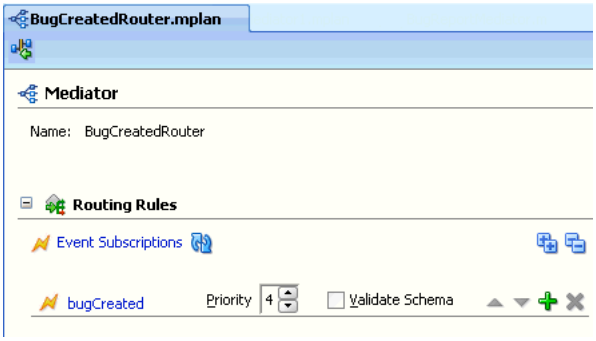
   For detailed procedures on creating SOA composite application projects, see the chapters "Developing SOA Composite Applications with Oracle SOA Suite" and "Getting Started with Oracle Mediator" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

   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 example application, a mediator service component named `BugCreatedRouter` is subscribed to the `BugCreated` event, as shown in Figure 32–2.

Figure 32–2 Mediator Subscription to an Event



6. Create the BPEL process service component.

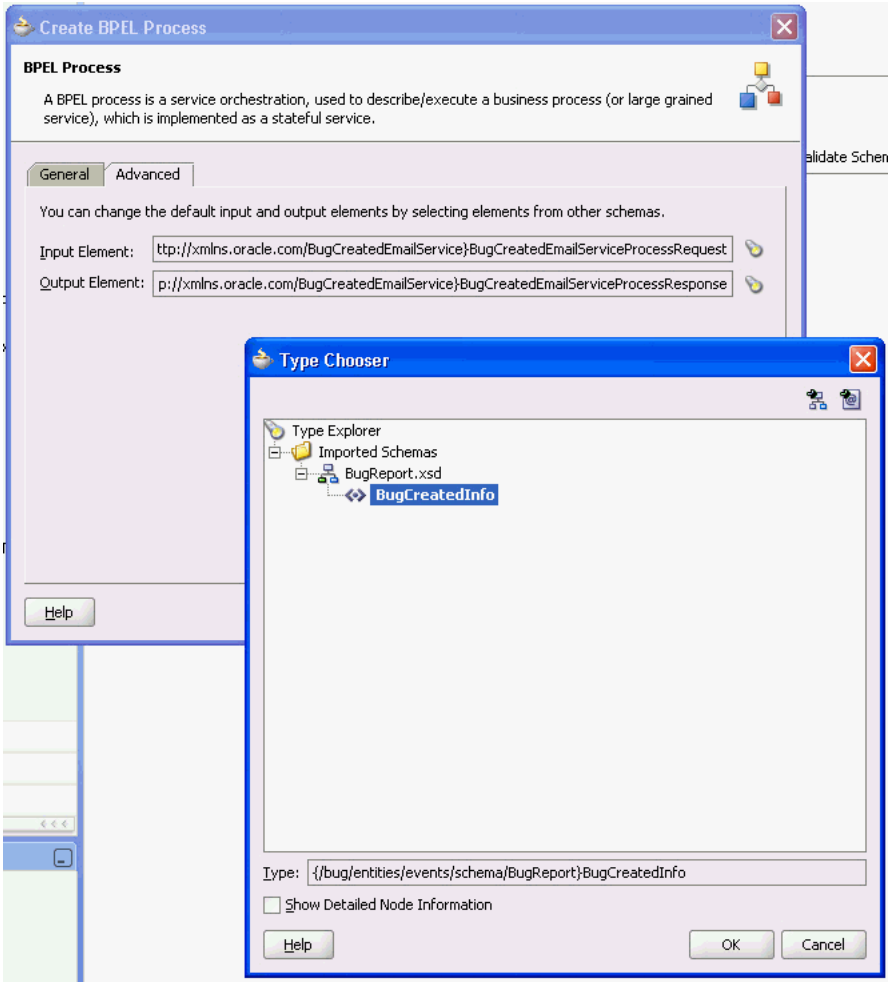
   For detailed procedures, see the chapter "Getting Started with Oracle BPEL Process Manager" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

   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 example application, the input element would be the `BugCreatedInfo` payload under the `BugReport.xsd` node, as shown in Figure 32–3.

Figure 32–3 Determining the Input Element for the Payload

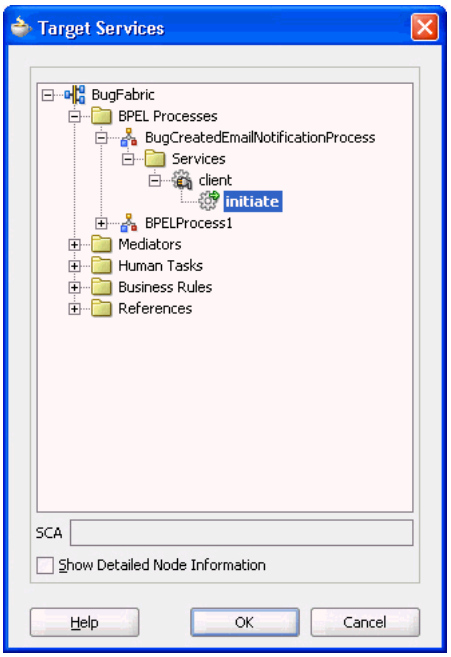


   - You create an activity that accepts the payload from the input element as its input parameters. In the sample application, an email activity is created that takes various input parameters from the payload as assigns them to the email's parameters.

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 32–4.

Figure 32–4 Target for a Routing Rule



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. For details, see the chapter "Deploying SOA Composite Applications" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.
10. Use Oracle Enterprise Manager Fusion Middleware Control Console to view the SOA composite application to ensure it was properly deployed. For more information about using Oracle Enterprise Manager Fusion Middleware Control Console, see the chapter "Deploying SOA Composite Applications" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

32.5 Alternative Approaches

Following are alternative approaches to the use case pattern:

- Section 32.5.1, "Using the Java Event API to Publish Events"
- Section 32.5.2, "Using a JAX-WS Proxy to Invoke a Synchronous BPEL Process"

32.5.1 Using the Java Event API to Publish Events

You can programmatically 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 Section 32.4, "How to Initiate a BPEL Process Service Component from an Oracle ADF 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 32–4.

Example 32–4 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;
```

2. Publish events as required. An example is shown in Example 32–5.

Example 32–5 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);

}
```

32.5.2 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.

For more information about this approach, see the chapter "Integrating Web Services Into an Oracle Fusion Web Application" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

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 32–6.

Example 32–6 Using WebServiceConnection

```
WebServiceConnection wsc =

(WebServiceConnection)ADFContext.getCurrent().getConnectionsContext().lookup(conne
ctionName);
 Hello hello = wsc.getJaxWSPort(Hello.class);
```

Use the username or SAML token policies for identity propagation and security. For more information, see Chapter 50, "Securing Web Services Use Cases."

32.6 Securing 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 Chapter 50, "Securing Web Services Use Cases."

32.6.1 Running 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 32–7 shows the composite subscription for the sample application.

Example 32–7 runAsRoles Attribute in a Composite Subscription

```
<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>
```

Note: Currently, the only option for `runAsRoles` is `$publisher`.

2. To validate the subject propagation in the SOA composite application, add the following code as shown in Example 32–8 to the BPEL file for the BPEL process flow:

Example 32–8 Using bplex:exec to Print Out Subject Information

```
<bpelx: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\n");
                System.out.println ("##########-----> 
                                              subject:  " + subject.toString());
   System.out.println("\n\n\n\n\n\n\n");

               ]]>
 </bpelx:exec>
```

32.6.2 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.

32.7 Verifying 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.

32.7.1 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 using various testing and debugging methods described in the chapter "Testing and Debugging ADF Components" in

the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. 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`

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.

32.7.2 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.

32.7.2.1 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 32–9 and Example 32–10.

In Example 32–9, an empty event with namespace `uuid:1111` and local name `MyEvent` is sent to the event bus.

Example 32–9 Sending an Empty Event

```
oracle.integration.platform.blocks.event.SendEvent -dbconn
host.us.oracle.com:1521:SID -dbuser user -dbpass password -eventName
{uuid:1111}MyEvent
```

In Example 32–10, the event contained in the file `AnEvent.xml` is sent to EDN.

Example 32–10 Sending an Event to EDN

```
oracle.integration.platform.blocks.event.SendEvent -dbconn
host.us.oracle.com:1521:SID -dbuser user -dbpass password -event AnEvent.xml
```

32.7.2.2 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 32–11 to the `JAVA_OPTIONS` environment variable.

Example 32–11 Setting the EDN Debug System Property

```
-Dedn.debug.event-connection=true
```

To access the EDN database log:

You can use the EDN database log shown in Example 32–12 for diagnostic purposes.

Example 32–12 EDN Database Log

```
http://HOSTNAME:7001/soa-infra/events/edn-db-log
```

32.8 Troubleshooting the Use Case

Following are tips that may help resolve common issues that arise when developing or running this use case.

32.8.1 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.

32.8.2 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`.

32.9 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).

32.10 Known Issues and Workarounds

Following are known issues:

- You must add a library reference to `oracle.soa.workflow.wc` in order to raise events successfully.
- You can only specify `$publisher` as the `runAsRole` for an event subscription.

33

Initiating 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.

33.1 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.

33.2 Other Approaches

Instead of using an event to invoke an Oracle Mediator service component from a PL/SQL stored procedure, you could use one of the following implementations.

- Invoke Oracle Mediator directly using UTL_HTTP or any other PL/SQL-based APIs.

> **WARNING: This approach is prohibited.**

33.3 Example

The sample code for this use case can be downloaded from Oracle SOA Suite samples.

33.4 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 33–1.

Example 33–1 Calling the publish_event Method

```
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=
   "http://oracle.com/fabric/businessEvent"
   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>
   <ReferedBy>Houston America Taxco</ReferedBy>
   </CU:CustomerData></eb:content></eb:business-event>');

  EDN_PUBLISH_EVENT( NAMESPACE => NAMESPACE, LOCAL_NAME => LOCAL_NAME, PAYLOAD =>
```

```
PAYLOAD);
END;
```

33.5 Securing the Design Pattern

Secure Oracle Mediator by configuring the property `runAsRoles=$publisher`. For details on securing the Oracle Mediator, see Section 32.6, "Securing the Design Pattern."

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.

To secure this pattern, follow the instructions described in Chapter 50, "Securing Web Services Use Cases."

33.6 Verifying the Deployment

Verifying the deployment involves the following:

- Section 33.6.1, "Testing and Deploying the Use Case"
- Section 33.6.2, "Verifying the SOA Composite Deployment Using Oracle Enterprise Manager Fusion Middleware Control Console"

33.6.1 Testing and Deploying the Use Case

Testing and deploying the use case involves the following main steps:

1. Test your Oracle ADF application using various testing and debugging methods described in the chapter "Testing and Debugging ADF Components" of the *Oracle Fusion Middleware Fusion Developer's Guide for 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
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:

   "Automating Testing SOA Composite Applications" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

33.6.2 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.

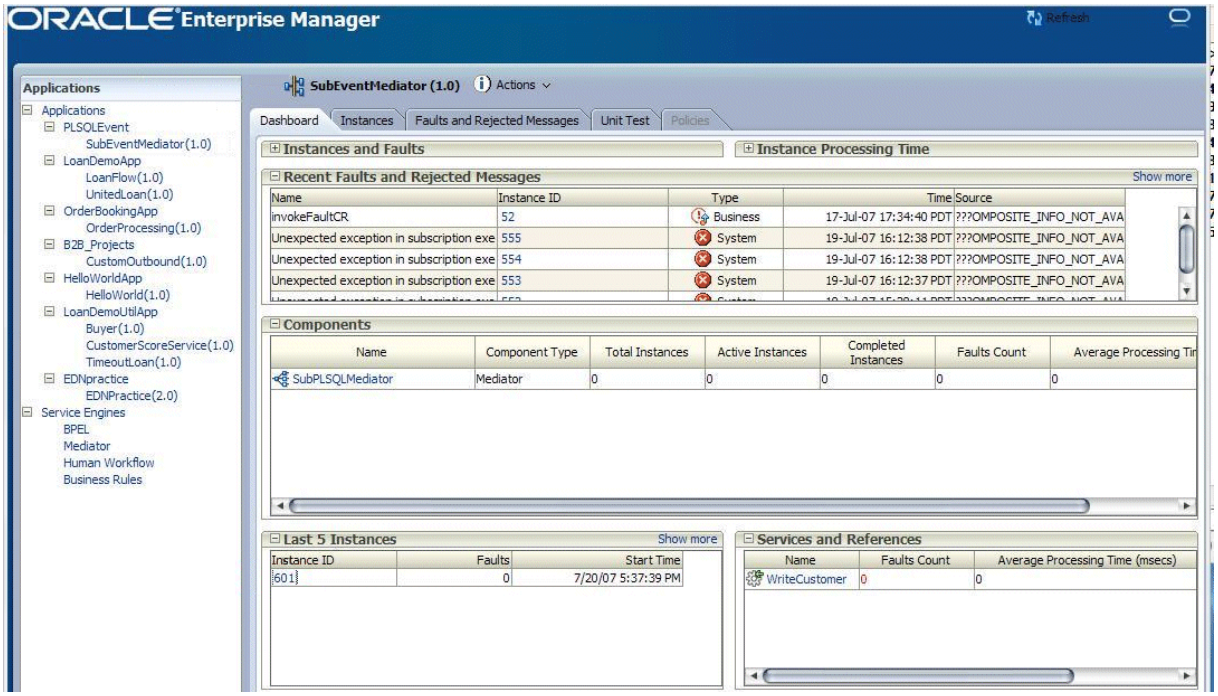
To verify that the SOA composite was successfully deployed and the event was received:

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
   ```

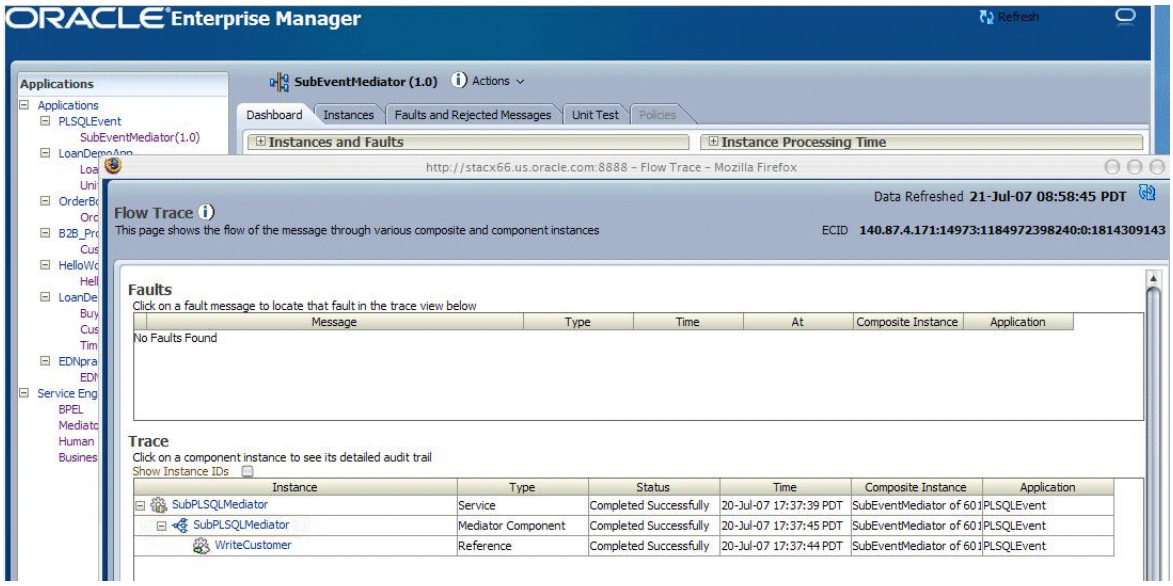
2. From the list of applications, expand the PLSQLEvent composite.
3. In the Last 5 Instances pane, click the most recent instance as shown in Figure 33–1.

Figure 33–1 Finding the Latest PLSQLEvent Composite Instance



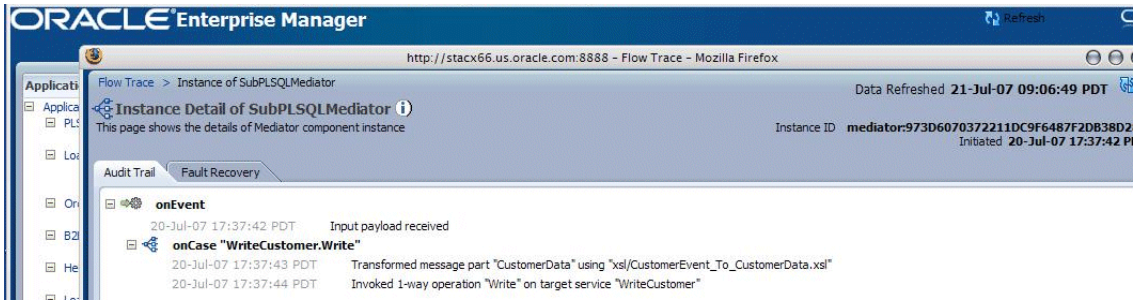
4. In the Flow Trace window that displays, click the Oracle Mediator component, as shown in Figure 33–2.

Figure 33–2 The Flow Trace Window



A window displays, showing the event results, as shown in Figure 33–3.

Figure 33–3 Displaying the Event



33.7 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`. See the troubleshooting section in the chapter "Deploying SOA Composite Applications" of the *Oracle SOA Suite Developer's Guide* for more information.
- 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`.

33.8 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 prior to implementing your own version of this use case. Running the sample ensures that the EDN database queue works as expected.

33.9 Known Issues and Workarounds

Following are known issues:

- Event publishing is an asynchronous action, there is no support for synchronous event publishing.

34

Orchestrating 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.

See Chapter 35, "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:

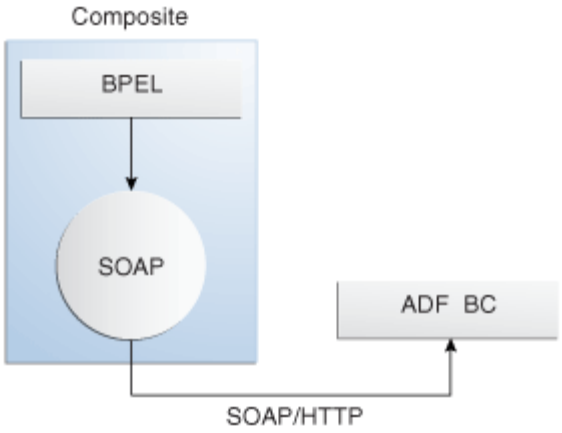
- SOA composite application that includes a BPEL process service component
- Oracle ADF web application that includes ADF Business Components

34.1 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 34–1 shows a high-level overview of this design pattern.

Figure 34–1 SOA Composite Application Accesses ADF Business Components Using SOAP



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.

34.2 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.)

34.3 Example

The sample code for this use case can be downloaded from Oracle SOA Suite samples.

34.4 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, as documented in "Part II: Building Your Business Services" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

2. Optionally, create any SDO classes for view objects. For detailed procedures, see the chapter "Integrating Service-Enabled Application Modules" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

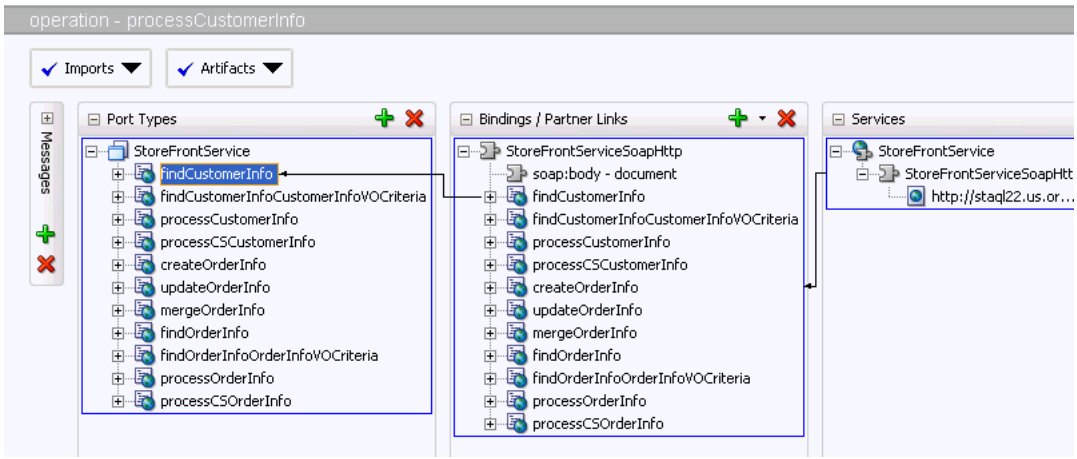
   Any top-level view objects referenced in a service interface created from an application module will automatically be service-enabled. However, creating the SDO classes for individual view and entity objects allows you to configure the SDO name or namespace, or selectively service-enable child view objects.

3. Configure the service interface for the application module. For detailed procedures, see the chapter "Integrating Service-Enabled Application Modules" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

   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 34–2 shows the WSDL created for the StoreFrontService in the sample application. This service accesses the customer and order information needed by the SOA composite application.

Figure 34–2 StoreFrontService WSDL File in Oracle JDeveloper



4. Implement the security as described in Section 34.5, "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. For details, see the chapter "Deploying Fusion Web Applications" of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

   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 34–1 to `oracle.adf.domain.loader`.

Example 34–1 Shared Library Name

```
<shared-library name="adf.oracle.domain" version="11.1.1"
                library-compatible="true">
...
<import-shared-library name="oracle.soa.workflow.wc"/>
```

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.

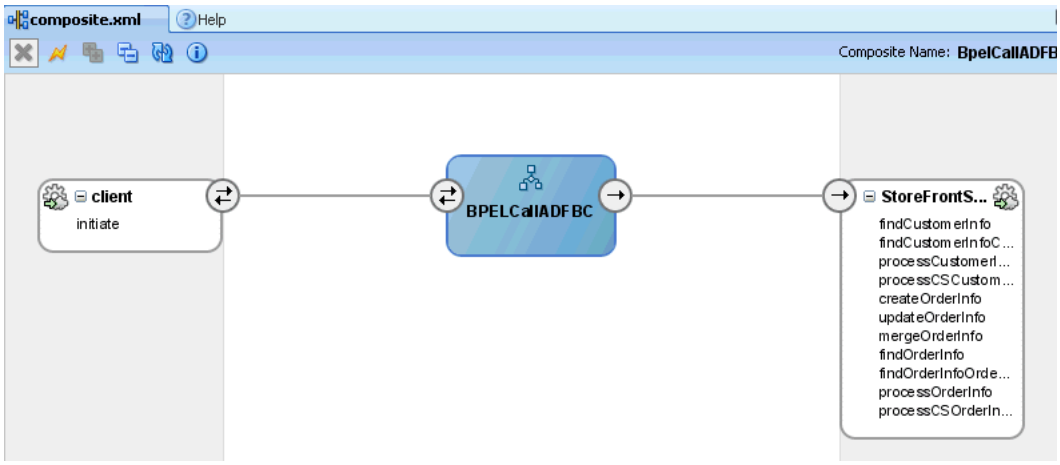
7. Create a SOA composite application that includes a BPEL process service component. For more information, see the chapter "Developing SOA Composite Applications with Oracle SOA Suite" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

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 34–3 shows the `composite.xml` file for the sample application.

Figure 34–3 Composite Application



A Partner Link for the ADF Business Components is automatically created when the reference is wired to the BPEL process.

For more information about creating bindings, see the chapter "Developing SOA Composite Applications with Oracle SOA Suite" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

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.

   For more information about wiring references, see the chapter "Developing SOA Composite Applications with Oracle SOA Suite" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

10. Create an Invoke activity that invokes the partner link for the ADF Business Components.

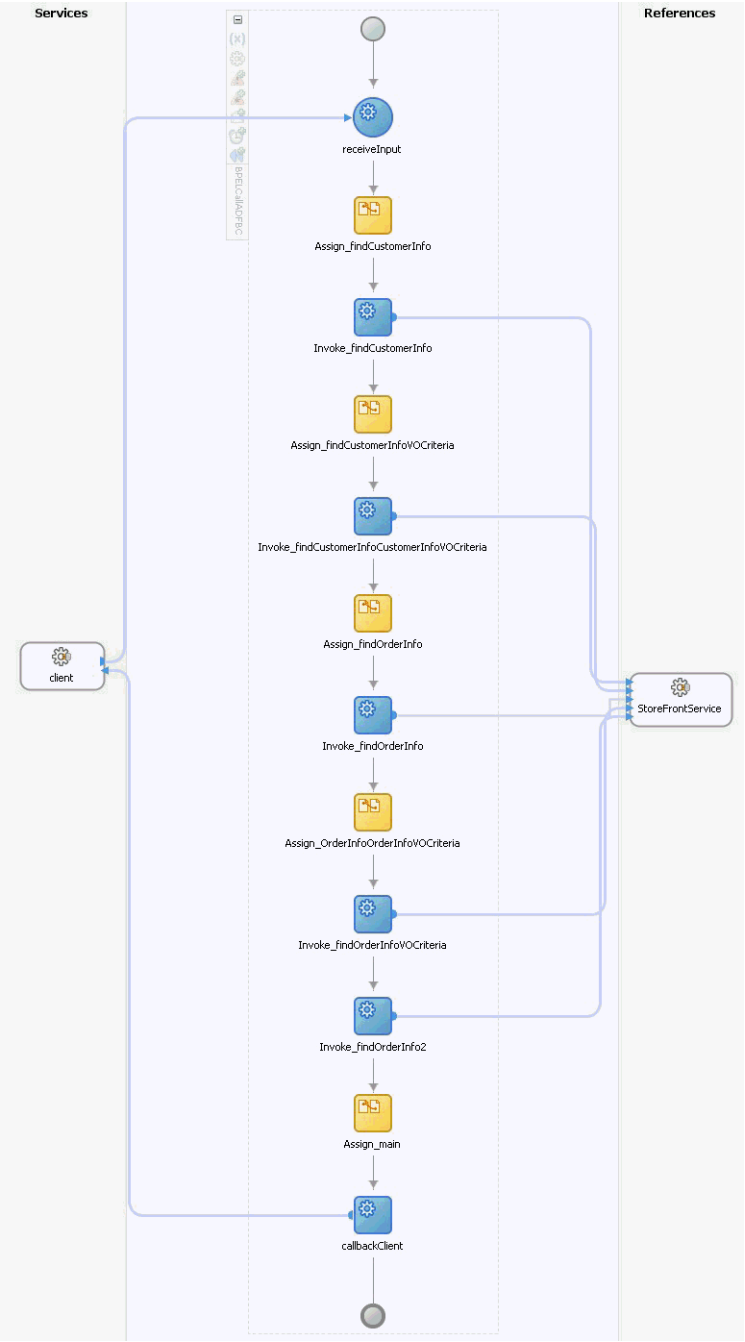
    For more information about creating Invoke activities, see the chapter "Getting Started with Oracle BPEL Process Manager" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

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.

    For more information about creating Assign activities, see the chapter "Getting Started with Oracle BPEL Process Manager" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

12. Repeat adding assign and invoke activities as needed. Figure 34–4 shows the BPEL flow for the sample application.

Figure 34–4 BPEL Flow for Invoking an ADF Business Components Service



13. Implement the security as described in Section 34.5, "Securing the Design Pattern".

34.5 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 Section 50, "Securing Web Services Use Cases."

34.6 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 using the various testing and debugging methods described in the chapter "Testing and Debugging ADF Components" of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. For information about testing the ADF Business Components service, see the chapter "Integrating Service-Enabled Application Modules" chapter of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
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. For more information, see "Managing SOA Composite Applications" in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

34.7 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. For more information, see the section "Automating Testing for SOA Composite Applications" in the chapter "Managing SOA Composite Applications" in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

34.8 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.

35

Manipulating Back-End Data from a SOA Composite

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.

35.1 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.

35.2 Example

The sample code for this use case can be downloaded from Oracle SOA Suite samples.

35.3 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, as documented in the chapter "Implementing Business Services with Application Modules" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
2. Optionally create any SDO classes for view objects. For detailed procedures, see the chapter "Integrating Service-Enabled Application Modules" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

   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. For detailed procedures, see the chapter "Integrating Service-Enabled Application Modules" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

   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 *name*`ServiceImpl` class and selecting **Run**. Alternatively, define an ADF Business Components service interface profile and deploy it to the standalone Oracle WebLogic Server. For more information, 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
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. For detailed procedures see the chapter "Developing SOA Composite Applications with Oracle SOA Suite" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.
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.

   For more information about creating bindings, see the section "Adding Service Binding Components" in the chapter "Developing SOA Composite Applications with Oracle SOA Suite" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite* and "Using ADF Model in a Fusion Web Application" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
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.

   For more information about creating entity variables, see the chapter "Manipulating XML Data in a BPEL Process" of the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*

   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 35–1 shows the XML for a bind activity that establishes `OrderId` as the key that will be passed to retrieve orders.

Example 35–1 Bind Activity Establishes the Key

```
<bpelx:bindEntity name="BindEntity_1" variable="orderEntityVar">
    <bpelx:key  keyname="ns4:OrderId">bpws:getVariableData('inputVariable',
    'payload','/client:BPELEntityVarADFBCProcessRequest/client:orderID')
    </bpelx:key>
</bpelx:bindEntity>
```

   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 *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite* and "Using ADF Model in a Fusion Web Application" in the *Oracle*

Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework.

10. Add an Assign activity to the BPEL process service component that will manipulate the data.

    Example 35–2 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 Section 35.7.2, "Support for XPath Operations."

Example 35–2 Assign Activity Manipulates an Entity Variable Value

```
<assign>
    <copy>
        <from expression="$orderEVar/fdsm:OrderTotal + 1" />
        <to variable="orderEVar" query="fdsm:OrderTotal" />
    </copy>
 </assign>
```

    Once 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 Section 35.7.1, "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:

    - "Using Oracle BPEL Process Manager Sensors" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*
    - "Monitoring BPEL Process Service Components and Engines" in the Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite
    - "Managing SOA Composite Applications" in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

11. Deploy the ADF Business Components web service to the Integrated Oracle WebLogic Server with SOA infrastructure. Then deploy the SOA composite application.

35.4 Securing the Design Pattern

To secure this pattern, it is recommended that you follow the same steps as described in Section 34.5, "Securing the Design Pattern."

35.5 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 using the various testing and debugging methods described in the chapter "Testing and Debugging ADF Components" of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. For information about testing the ADF Business Components service, see the chapter "Integrating Service-Enabled Application Modules" chapter of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
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. For more information, see "Managing SOA Composite Applications" in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

> **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.

35.6 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`.

35.7 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.

35.7.1 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 as a result of 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 35–3 includes a locally declared entity variable.

Example 35–3 Local Entity Variable

```
<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>
```

Because the variable was defined locally within the scope `myScope`, once 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 35–4 shows an entity variable defined globally, hence its scope will complete when the instance completes.

Example 35–4 Global Entity Variable

```
<process ...>
  <variables>
           <variable name="orderEVar" element="fdsm:orderInfo2SDO"
                                      bpelx:entity.si="OrderService"  />
  </variables>
  <sequence>
    <receive ...>
      <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 ...>
  </sequence>
</process>
```

35.7.2 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 35–5 illustrates the verification of variable data integrity.

Example 35–5 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 "/>
```

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.

35.7.3 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 Chapter 34, "Orchestrating ADF Business Components Services" can be easily combined.

36

Accessing a PL/SQL Service from a SOA Composite

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.

36.1 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.

36.2 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.

36.3 Example

The sample code for this use case can be downloaded from Oracle SOA Suite samples.

36.4 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, as documented in the section "Part II: Building Your Business Services" of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
2. Write a method in your application module that accesses the PL/SQL stored procedure. For step-by-step instructions, see the section "Invoking Stored Procedures and Functions" in the chapter "Advanced Business Components Techniques" of the*Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
3. Generate the service interface for the business component, as described in "Integrating Web Services Into a Fusion Web Application" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
4. Invoke it through a SOAP binding, as described in Chapter 34, "Orchestrating ADF Business Components Services."

36.5 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 Section 34.5, "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 Section 47, "Implementing Application User Sessions."

For information about securing the design pattern, see Section 50, "Securing Web Services Use Cases."

36.6 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 using various testing and debugging methods described in the chapter "Testing and Debugging ADF Components" of the *Oracle Fusion Middleware Fusion Developer's Guide for 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

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:

   "Automating Testing SOA Composite Applications" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

37

Invoking Custom Java Code from a SOA Composite

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.

37.1 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.

37.2 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 `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.

For more information about using the `bpelx:exec` command, see the chapter "Incorporating Java and Java EE Code in a BPEL Process" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

Another alternative is to call custom Java classes using custom XPath functions. For more information about using XPath functions to call a Java class, see the sub-section "Creating User-Defined XPath Extension Functions" in the appendix "XPath Extension Functions" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

37.3 Example

Download the sample code for this use case from the following location. Oracle SOA Suite samples.

37.4 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, as documented in the section "Part II: Building Your Business Services" of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
2. Generate the service interface for the business component and from the SOA component, invoke the service through a SOAP binding, as described in Chapter 34, "Orchestrating ADF Business Components Services."

37.5 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 Section 34.5, "Securing the Design Pattern."

37.6 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 using various testing and debugging methods described in the chapter "Testing and Debugging ADF Components" of the *Oracle Fusion Middleware Fusion Developer's Guide for 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
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:

   "Automating Testing SOA Composite Applications" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

37.7 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.

For more information about testing and debugging, see the following chapter:

"Testing and Debugging ADF Components" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

37.8 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.

38

Managing Tasks from an Oracle ADF Application

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 needs to be raised to the employee's manager so that he or she 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. Once 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.

38.1 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.

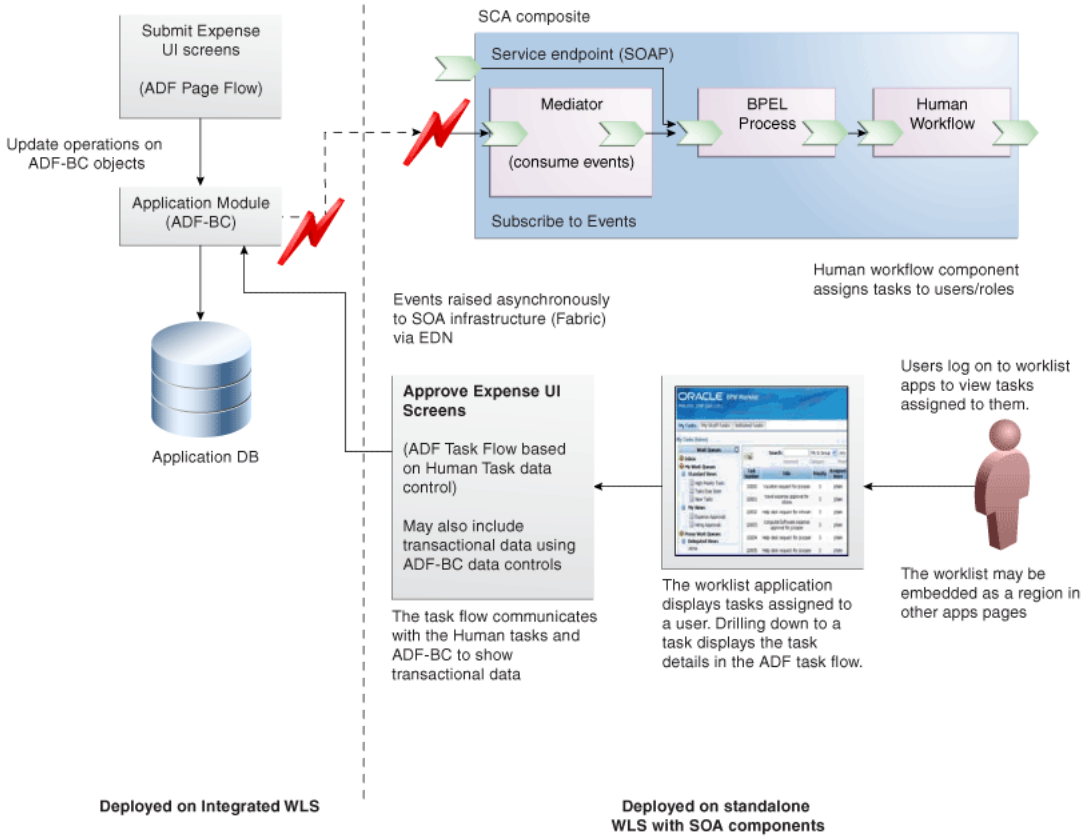
To develop this pattern, an ADF application can invoke an SOA composite application, as described in Chapter 32, "Initiating a SOA Composite from an Oracle ADF Web Application." A human task service component can be included in the composite that assigns tasks to users or roles. This component includes 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 38–1 shows the recommended pattern.

Figure 38–1 Recommended Pattern



This chapter explains how to implement the recommended pattern.

38.2 Other Approaches

There are no other supported approaches to this use case.

38.3 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 needs to 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.

38.4 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 Section 32.4, "How to Initiate a BPEL Process Service Component from an Oracle ADF 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 Section 32.4, "How to Initiate a BPEL Process Service Component from an Oracle ADF 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.

   For detailed procedures on creating a human task service, see the chapter "Designing Human Tasks" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

   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 Section 38.6, "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. For more information about notifying users of changes to task status, see the chapter "Designing Human Tasks" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.
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.

   Once 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 Chapter 35, "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.

   For detailed procedures, see the chapter "Designing Human Tasks" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

   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.

38.5 Other Approaches

You can deploy the human task flow to a remote server without SOA infrastructure.

For more information, 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 *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

38.6 Securing the Design Pattern

> **Note:** In order 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 38–1.

Example 38–1 Identity Service Configuration

```
<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>
```

Fore more information regarding 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 *Oracle Fusion Middleware Administrator's Guide for 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 Application Security Guide*.

> **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.

38.7 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 for the Worklist application, and then run the ADF application.

To verify this design pattern:

1. Test your Oracle ADF web application using the various testing and debugging methods described in the chapter "Testing and Debugging ADF Components" of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application*

Development Framework. 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. For more information about using Fusion Middleware Control Console, see the chapter "Deploying SOA Composite Applications" of the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.
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:

   `http://localhost:8888/fabric/OrderBookingApp/OrderProcessing/Client`
4. View the ADF Business Components runtime log to view the event and payload. For more information about the log, see the chapter "Testing and Debugging ADF Components" of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
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.

   For more information about Fusion Middleware Control Console, see the chapter "Deploying SOA Composite Applications" of the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.
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.

38.8 Troubleshooting the Use Case

Following are tips that may help resolve common issues that arise when developing or running this use case.

38.8.1 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 Section 38.8.3, "Logging").
  - `MW_HOME/user_projects/domains/<domain name>/config/config.xml`
  - `MW_HOME/user_projects/domains/soainfra/logs/startsoa.log`
  - `MW_HOME/user_projects/domains/soainfra/servers/AdminServer/tmp/_WL_user/soa-infra/77op2b/war/WEB-INF/application.log`

  Any exception where the exception stack trace has classes from package `oracle/bpel/services/workflow` indicates a workflow exception.
- 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.

38.8.2 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 Section 38.8.3, "Logging." Set the log level to FINE and deploy the task flow again. Once you do this, you should see error messages in the log file.
  - If you get any errors on the task details screen when performing any operations, check the following log files:
    * `MW_HOME/user_projects/domains/DOMAIN_NAME/servers/ADMIN_SERVER_NAME/logs/DOMAIN_NAME.logl`
    * `MW_HOME/user_projects/domains/DOMAIN_NAME/servers/SERVER_NAME/logs/SERVER_NAME.log`

38.8.3 Logging

You can set logging for the Workflow application and for the ADF task flow.

38.8.3.1 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 38–2. 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 38–2 Adding a New Log Handler

```
<?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>
```

38.8.3.2 ADF Task Flow Logging

To enable ADF logging for the task flow, add fragments shown in Example 38–3 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 38–3 Adding Fragments to the Logging File

```
<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>
```

38.9 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.

For more information, please refer to the chapter "Managing SOA Composite Applications" in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

39

Working with Data from a Remote ADF Business Components Service

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.

39.1 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.

39.2 Potential Approaches

If the data that you need to work with is always local to you, that is, available in the same database, then a table-based entity object should be used 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 Chapter 41, "Synchronously Invoking an ADF Business Components Service from an Oracle ADF Application." Since 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.

39.3 Example

Currently, no example is available.

39.4 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**. For more information see the section "Accessing Remote Data Over the Service-Enabled Application Module" in the chapter "Integrating Service-Enabled Application Modules" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

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. For more information, see the chapter "Defining SQL Queries Using View Objects" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
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 39–1 shows sample code used in the `connections.xml` file.

Example 39–1 Sample connections.xml Code

```
<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>
        <StringRefAddr addrType="jndiFactoryInitial">
           <Contents>weblogic.jndi.WLInitialContextFactory</Contents>
        </StringRefAddr>
        <StringRefAddr addrType="jndiProviderURL">
           <Contents>t3://localhost:7101</Contents>
        </StringRefAddr>
     </RefAddresses>
  </Reference>
```

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.

39.5 Securing the Design Pattern

For more information about securing the use case, see Chapter 50, "Securing Web Services Use Cases."

39.6 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 using the various testing and debugging methods described in the chapter "Testing and Debugging ADF Components" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

39.7 Troubleshooting the Use Case

Currently, there are no tips for troubleshooting the use case.

39.8 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.

39.9 Known Issues and Workarounds

Known limitations are the following:

- A service-based entity object cannot be referenced as an 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.

40

Invoking an Asynchronous Service from a SOA Composite

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.

40.1 Introduction to the Recommended Design Pattern

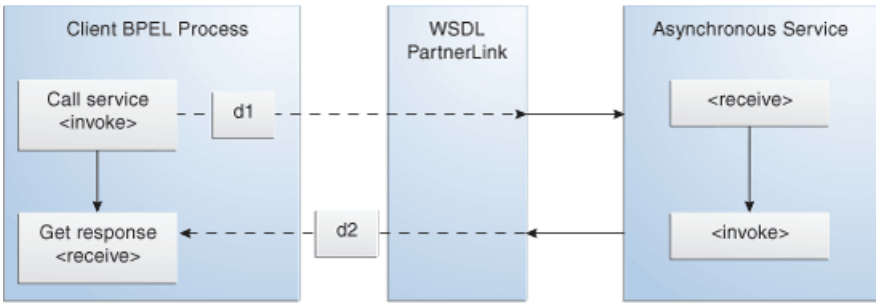
Oracle Fusion web applications often include services with long-running computation that are exposed through service interfaces. When invoking services through a synchronous service interface, the service may not execute as desired due to time outs or lack of resources as a result of blocked, waiting threads. To solve this problem, service 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 40–1 illustrates the process flow of BPEL process invoking an asynchronous service.

Figure 40–1 Asynchronous Service Call Flow



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.

40.2 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.

40.3 Example

An interface table is populated by third-party entities using legacy interfaces, such as FTP and files. Once 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 here:

Oracle SOA Suite samples

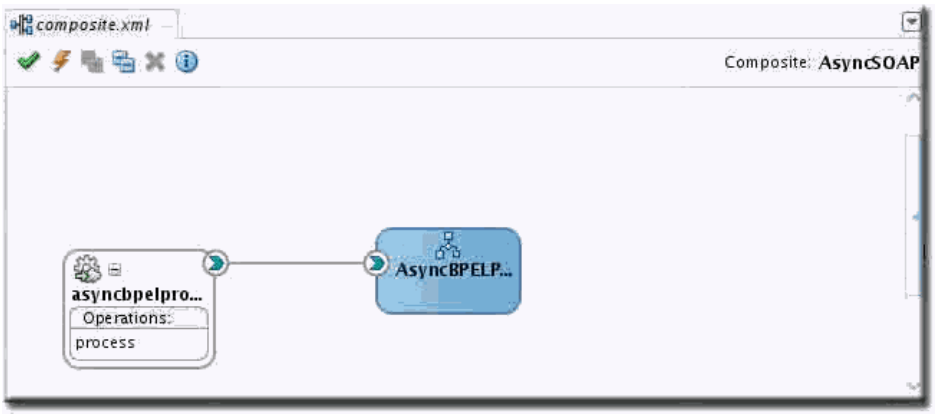
40.4 How to Invoke a SOA Composite Application from Within a SOA Composite Application

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 40–2 shows an example of a minimalist composite with a new BPEL process.

Figure 40–2 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.

40.4.1 Defining 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.

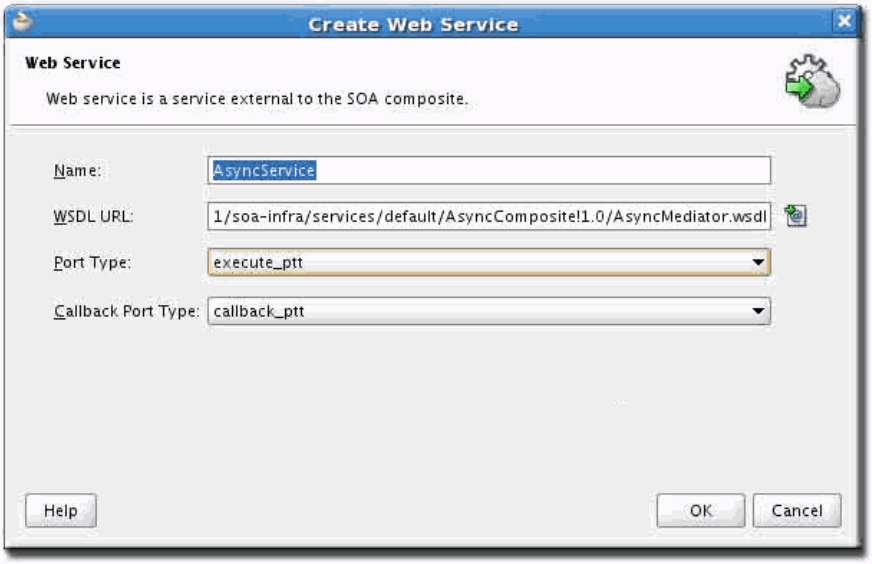
   The Create Web Service dialog opens.

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 40–3 shows an example of a completed Create Web Service dialog.

Figure 40–3 Create Web Service Dialog



40.4.2 Wiring the BPEL Process to the New Web Service Reference

In order 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.

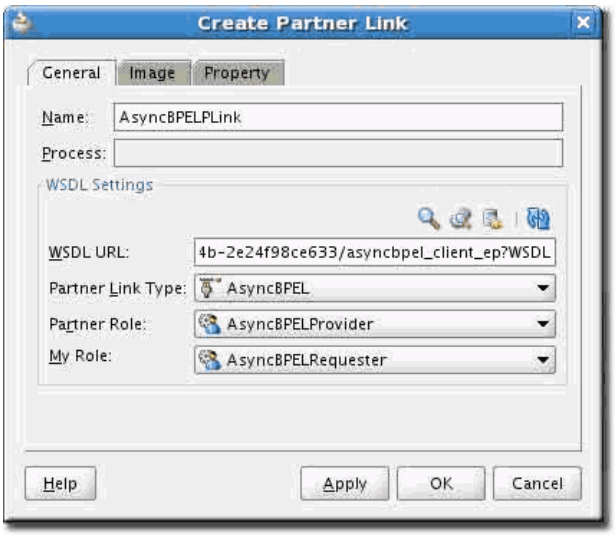
To define a partner link in the BPEL process editor:

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.

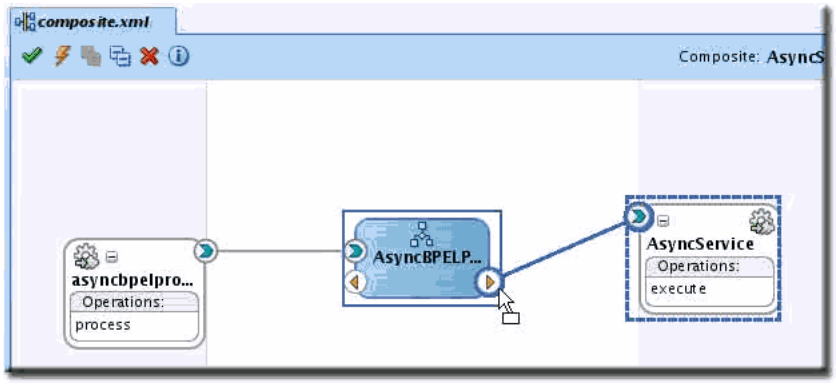
   The Create Partner Link dialog displays, as shown in Figure 40–4.

Figure 40–4 Create Partner Link Dialog



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 *`ServiceName`*`Provider` role. This is the role of the producer service for which you entered the WSDL.
   - **My Role:** From the dropdown list, select the *`ServiceName`*`Requester` 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 40–5.

Figure 40–5 Dragging the Interface Pin



40.4.3 Invoking 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:

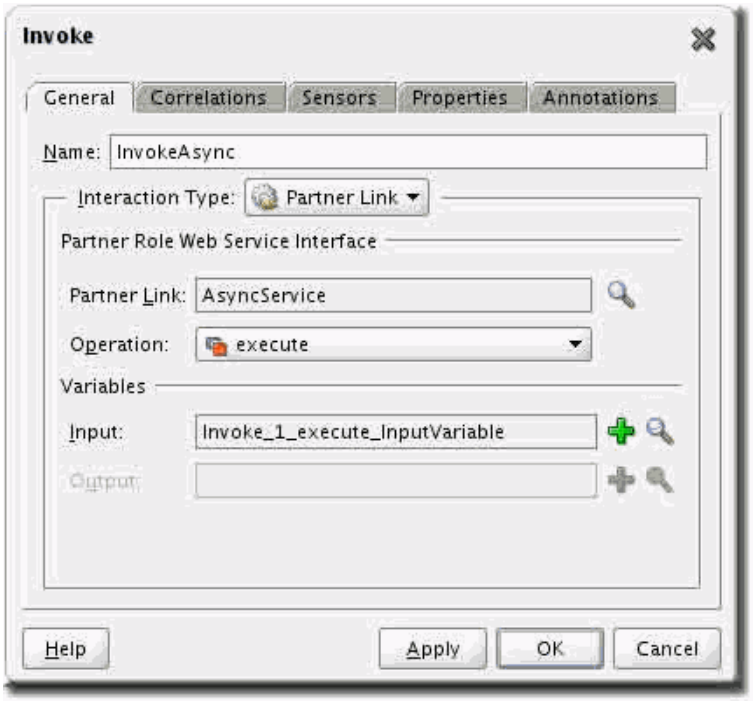
1. To adhere to Oracle Fusion Applications fault handling and logging standards, add the following activities:
   - **Scope:** By default, the main BPEL flow is the top-level scope of the process. However, should any of the activities generate a fault within that scope, compensation or recovery options are limited. Leverage nested scopes in order 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 in order 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 40–6 shows a completed Invoke Activity dialog.

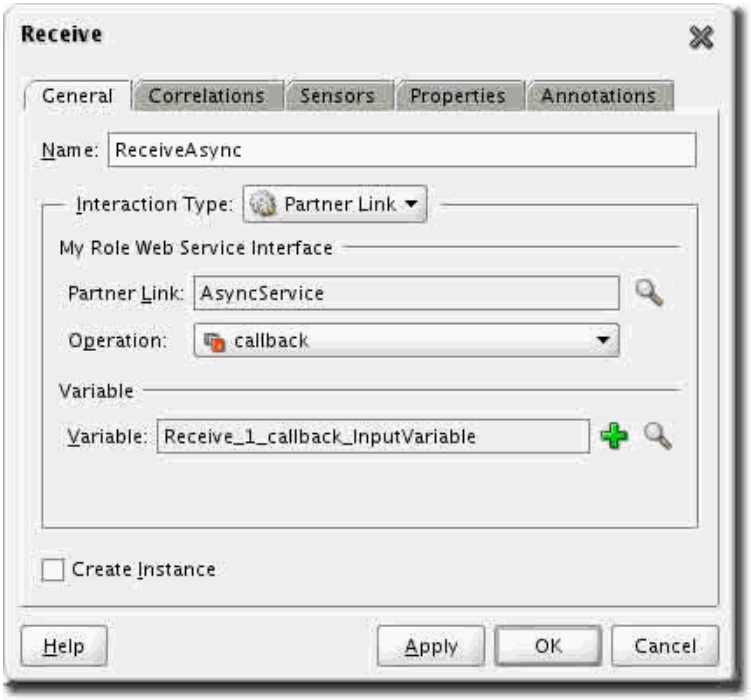
Figure 40–6 Invoke Activity Dialog



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 40–7 shows a completed Receive Activity dialog.

Figure 40–7 Receive Activity Dialog

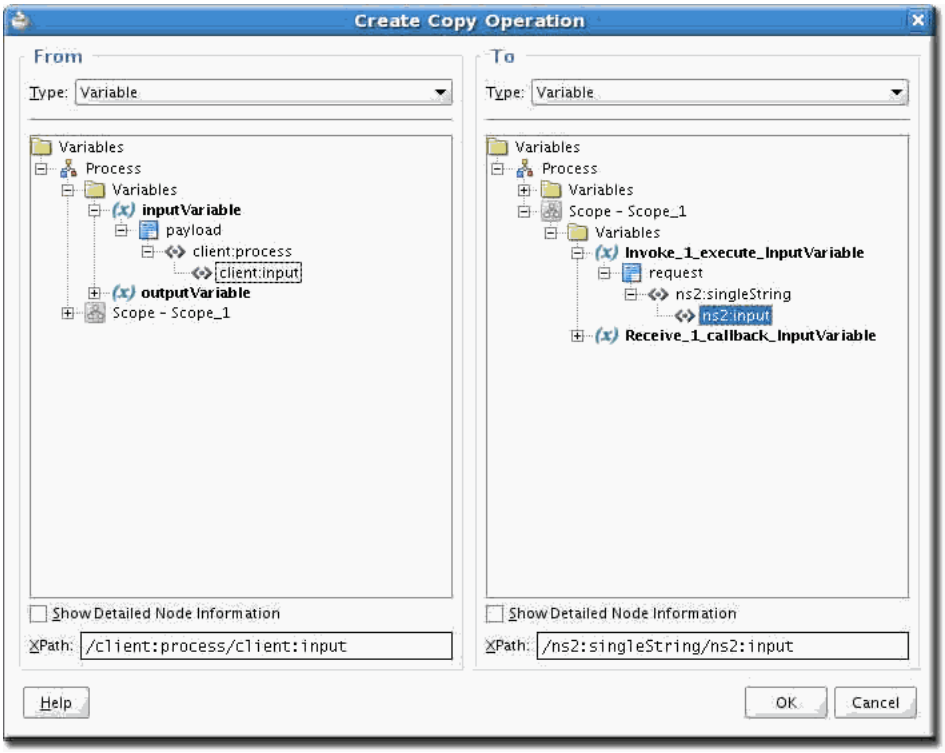


9. Drag an assign activity onto the flow (within the scope) just above the invoke activity you created earlier. Drag a second assign activity onto the flow just below the receive activity you created earlier.
10. Name each assign activity.

    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.

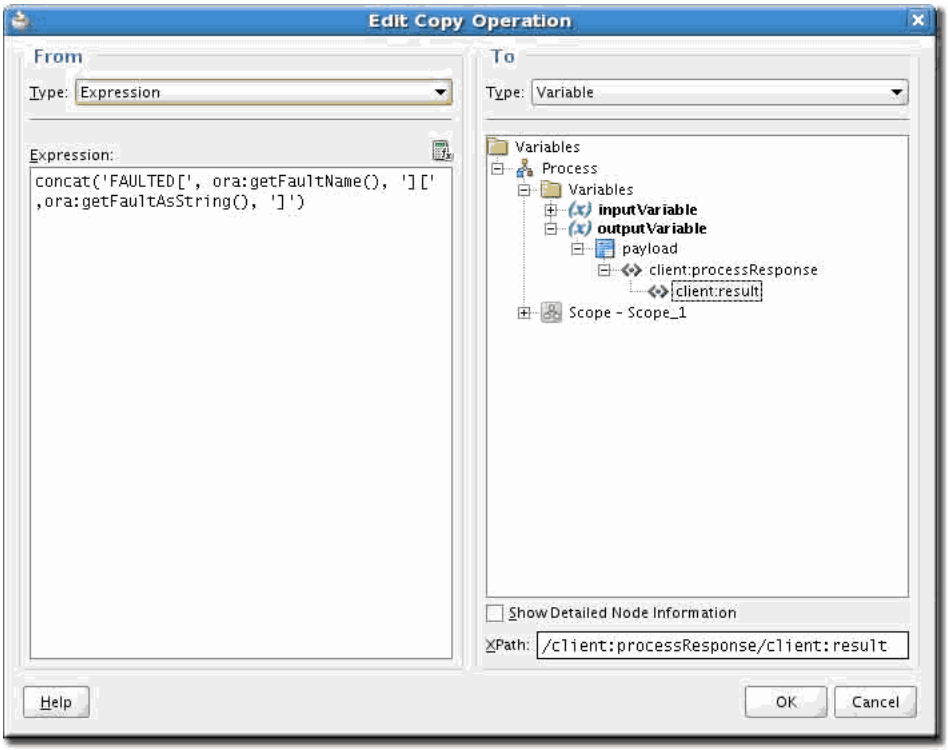
    In the Create Copy Operation dialog, select **Copy Operation**.
12. Depending on the type of data you want to assign to the invoke input variable, configure the following.
    - **Variable:** Select the contents of an element from the source variable, such as the default BPEL *inputVariable*, as shown in Figure 40–8.

Figure 40–8 Variable Copy Operation



- **XPath Expression:** Enter an XPath expression as required, such as *concat($variable1, '-', $variable2)*, as shown in Figure 40–9.

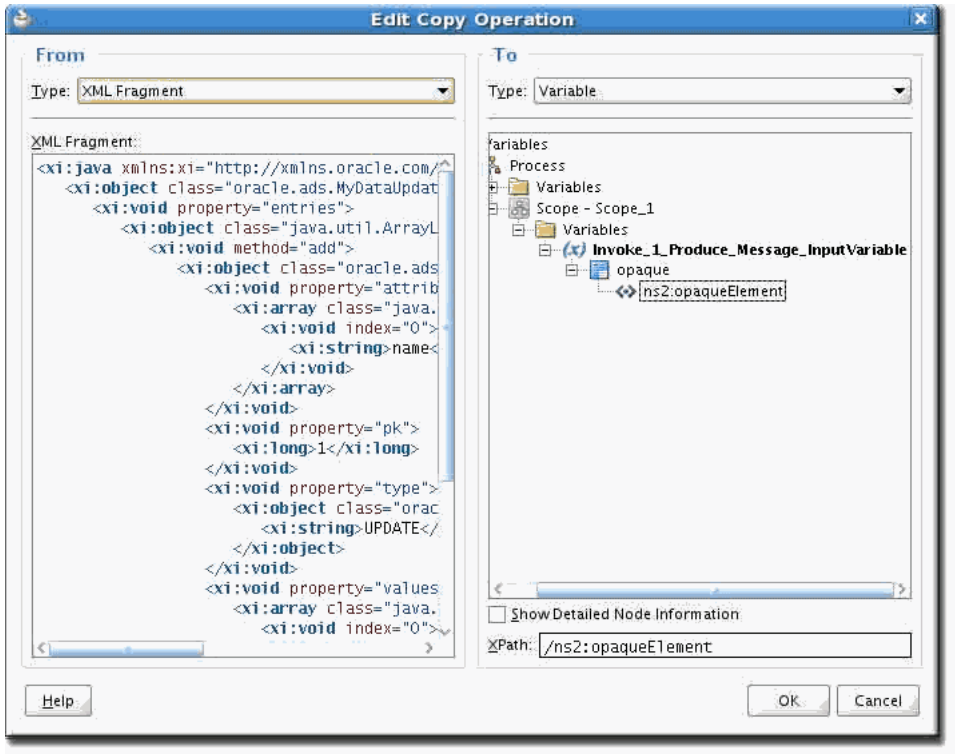
Figure 40–9 XPath Expression Copy Operation



- **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 40–10.

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 40–10 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. For more information, see "Deploying SOA Composite Applications" in *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.
15. Test the application. For more information, see the chapter "Automating Testing of SOA Composite Applications" in *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

40.4.4 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.

40.4.5 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.

40.5 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 Chapter 50, "Securing Web Services Use Cases."

40.6 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.

40.7 Troubleshooting the Use Case

Following are tips that may help resolve common issues that arise when deploying or running this use case.

40.7.1 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.

40.7.2 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 Section 40.6, "Verifying the Deployment."

40.8 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.

41

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

41.1 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.

41.2 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 Chapter 39, "Working with Data from a Remote ADF Business Components Service."

41.3 Example

Currently, no example is available.

41.4 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. For more information, see the chapter "Integrating Service-Enabled Application Modules" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

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 41–1.

Example 41–1 Sample Code for the File connections.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>
         <StringRefAddr addrType="jndiFactoryInitial">
```

```
            <Contents>weblogic.jndi.WLInitialContextFactory</Contents>
          </StringRefAddr>
          <StringRefAddr addrType="jndiProviderURL">
            <Contents>t3://localhost:7101</Contents>
          </StringRefAddr>
        </RefAddresses>
      </Reference>
```

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.

   For more information, see the chapter "Integrating Service-Enabled Application Modules" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

4. In your ADF Business Components service or managed Java bean class, invoke the desired service using `ServiceFactory`, as shown in Example 41–2.

Example 41–2 ServiceFactory Code

```
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");
        org1.setDescription("Your org Description"); //... and set more attributes
        orgs.add(org1);
        svc.processOrganizatiion("Merge", orgs, null);
```

Sample queries are shown in Example 41–3.

Example 41–3 Query Samples

```
// 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);
        fc.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();
fc.setExcludeAttribute(true);
l.add("PurchaseOrderLine");
fc.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.
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);
fc.setChildFindCriteria(cfcl);
DeptResult dres = svc.findDept(fc, null);
pw.println("### Dept 10 and 2nd Emp ###");
```

41.5 Securing the Design Pattern

For more information about securing the use case, see Chapter 50, "Securing Web Services Use Cases."

41.6 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 using the various testing and debugging methods described in the chapter "Testing and Debugging ADF Components" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

42

Implementing 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.

Notes:

- 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

42.1 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.

42.2 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.

42.3 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, needs to 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.

42.4 How to Implement an Asynchronous Service Initiation with Dynamic UI Update

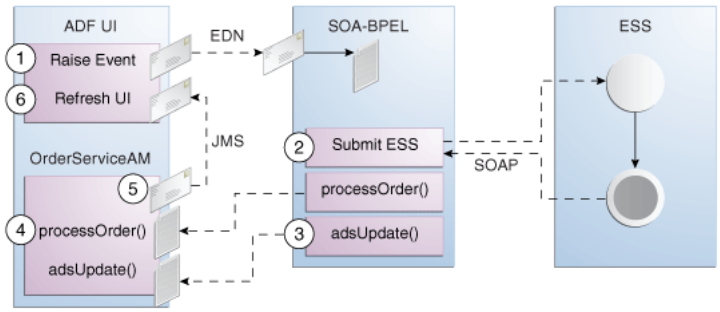
To enable the UI for dynamic update via ADS, you must first create the ADS handler, which uses a common set of JMS Queue handlers to broker the updates coming from the call to ADF Business Components services.

The main steps are as follows, as shown in Figure 42–1:

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 42–1 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 once 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**.

42.4.1 Writing the Active Data Handler

The classes shown in Example 42–1, Example 42–2, and Example 42–3 are common to every implementation of this pattern, and are responsible for handling JMS integration with ADS supported events.

Example 42–1 DemoDataChangeEntry.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,

        /**
         * 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) {
        this._values = _values;
    }
}
```

Example 42–2 DemoDataUpdateEvent.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 42–3 JMSHelper.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();
    }

    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();
      qsession.close();
      qcon.close();
    }

}
```

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`.
  2. Convert `DataUpdateEvent` to `ActiveDataEvent`. so that ADS can process the event.
  3. Deliver `ActiveDataEvent` to ADS.

Example 42–4 shows a collection model returned by a tree binding.

Example 42–4 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);
        }
        return cm;
    }
...
```

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 at this time, 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 42–5 shows sample `ActiveDataCollectionHandler` code.

Example 42–5 From the ActiveDataCollectionHandler Code

```
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;
   }
```

42.4.2 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 42–6 extends the Oracle ADF class `oracle.adf.view.rich.event.ActiveDataEntry` and implements several methods in that interface.

Example 42–6 Active Data Entry Class

```
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 ChangeType _changeType = null;
    private Object[] _path = null;
    private Object[] _insertPath = null;

}
```

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 42–7.

Example 42–7 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;
    }

    private List<ActiveDataEntry> _changeList =  Collections.emptyList();
    private int _eventId = 0;
}
```

42.4.3 Registering the Active Data Collection Model with the Oracle ADF UI Page

In order 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. As long as 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 42–8.

Example 42–8 adsBean Managed JavaBean

```
<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>
```

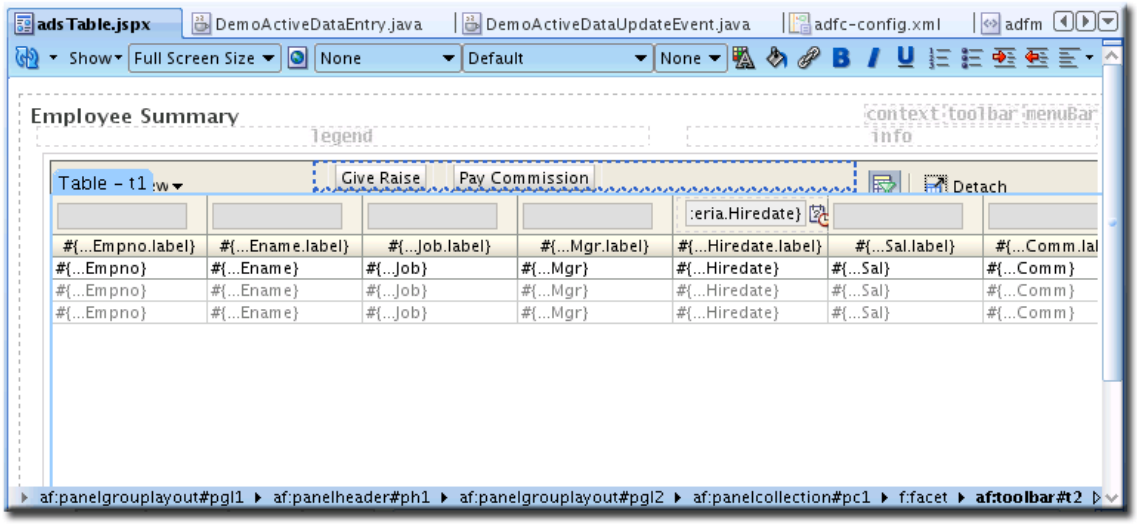
42.4.4 Registering the Component Managed JavaBean for Supporting Method Actions

To trigger the synchronous functionality of the use case pattern, raise a business event in response to the click of an Oracle ADF button. In order 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 42–2.

Figure 42–2 Dynamically Updated Table



The table component requires the managed JavaBean shown in Example 42–9.

Example 42–9 Table Component Managed JavaBean

```
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();
            // addConfirmationMessage();
        } catch ( Exception e ) {
            log.severe("ASM: Failed to raise commission event for key: " + rowKeyString);
        } // try
    }

    // 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();
            // addConfirmationMessage();
        } catch ( Exception e ) {
            log.severe("ASM: Failed to raise commission event for key: " + rowKeyString);
        } // try
    }
    // Invoke BPEL from here.


private void addConfirmationMessage() {
    FacesMessage msg = new FacesMessage("You request is submitted for approval.");
```

```
        FacesContext.getCurrentInstance().addMessage(null, msg);
    }
}
```

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 42–10.

Example 42–10 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>
```

42.4.5 Referencing the Managed JavaBean in the Page UI

Modify the page component to reference the managed JavaBean from the earlier steps, as shown in Example 42–11.

> **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 42–11 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}">
```

42.4.6 Creating the Data Model and Adding Application Module Methods

The data model should exist before the page is built in order 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.

For more information about creating a data model with application modules, see the chapter "Implementing Business Services with Application Modules" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

.

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 `payCommision` are supported for one or more selected employees. Use methods with simple string interfaces invoked by BPEL to complete the `payRaise` or `payCommision` 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 42–12.

Example 42–12 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
```

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 `payComission`. 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 42–13 shows a simplified event definition, while Example 42–14 shows an event schema definition.

Example 42–13 Simplified Event Definition

```
<?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 42–14 Event Schema Definition

```
<?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>
```

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.

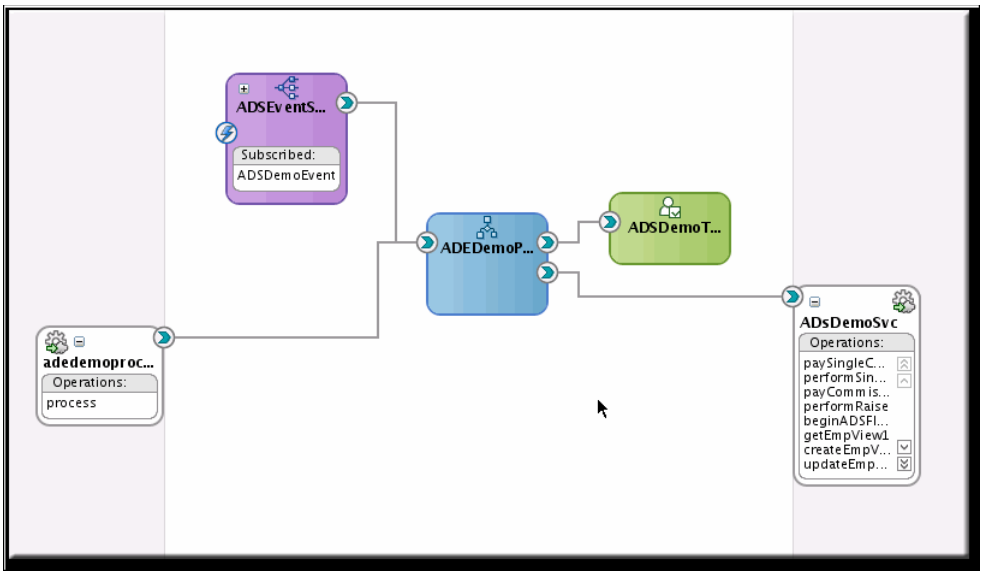
For more information about extending the application module with the `publishEvent` method, see Section 32.5.1, "Using the Java Event API to Publish Events."

Note: It is critical that the event name and namespace are consistent throughout the code and metadata definitions in the subscribing SOA composite.

42.4.7 Creating a SOA Composite that Subscribes to the Published Event

The creation of a SOA composite that subscribes to an event is covered in Section 32, "Initiating a SOA Composite from an Oracle ADF Web Application." A sample pattern composite is shown in Figure 42–3.

Figure 42–3 Pattern Composite

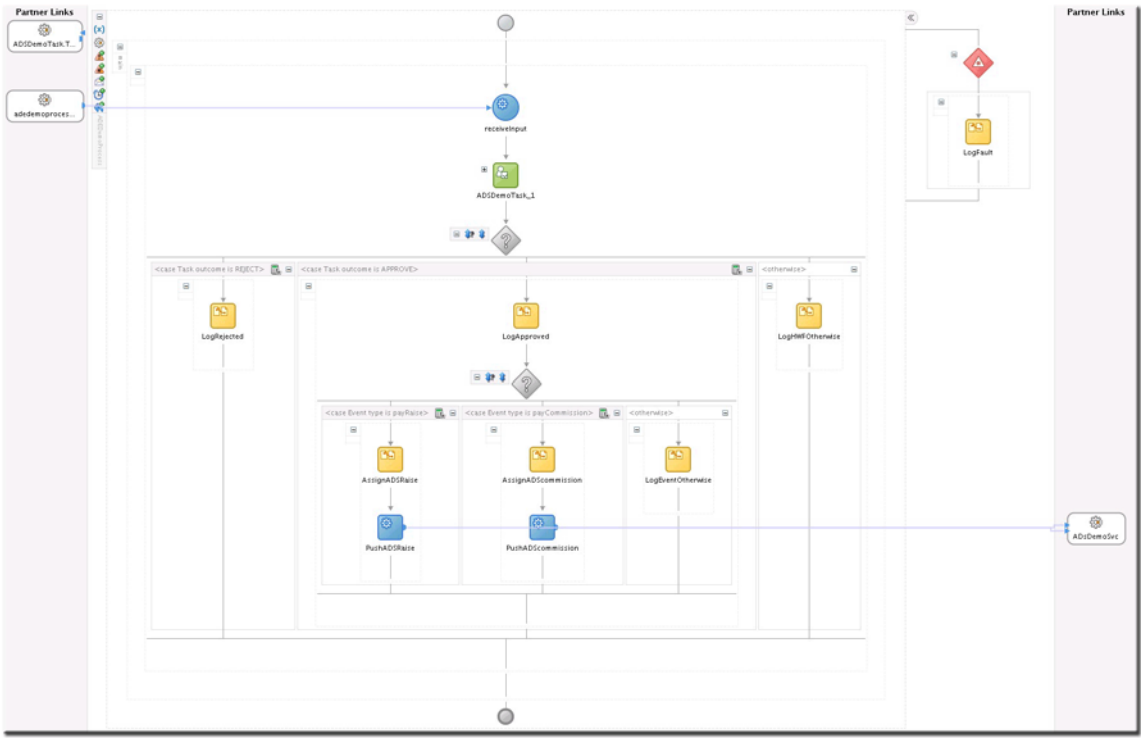


42.4.8 Constructing a BPEL Process to Perform Asynchronous Work

The creation of a BPEL process and human task activities is described in other sections. For more information, see Chapter 38, "Managing Tasks from an Oracle ADF Application."

A sample BPEL process is shown in Figure 42–4.

Figure 42–4 BPEL Flow



42.4.9 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 Chapter 34, "Orchestrating ADF Business Components Services."

42.5 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 Chapter 50, "Securing Web Services Use Cases."

42.6 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 42–15.

Example 42–15 Event Payload

```
Enqueing event: http://xmlns.oracle.com/apps/ta/adsdemo/events/edl::ADSDemoEvent from J
```

```
Body: <business-event xmlns:ns="http://xmlns.oracle.com/apps/ta/adsdemo/events/edl"
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:
Body: <business-event xmlns:ns="http://xmlns.oracle.com/apps/ta/adsdemo/events/edl"
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>
```

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=0000I2pqzVCBLA5xrOI7SY19uEYF00004g: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.adedemoprocess_
client.process
Apr 17, 2009 1:57:26 PM oracle.tip.mediator.dispatch.InitialMessageDispatcher
processCase
INFO: Immediate case {ADEDemoProcess.adedemoprocess_client.process} with case
id :{5B52B4A02B9211DEAF64D3EF6E2FB21D} 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.

42.7 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.

42.8 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

42.9 Known Issues and Workarounds

Known issues are as follows:

- Sparkle does not occur on selected rows.

43

Managing 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

43.1 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.

43.2 Potential Approaches

- Supported approach: Managing Tasks from an Oracle ADF application.
- Unsupported approach: Direct invoking a service using JAX-WS proxies.

43.3 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.

43.4 Managing 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.

43.4.1 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. In order 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.

43.4.2 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. Once 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.

43.4.2.1 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.

43.4.2.2 Import Code Packages into the Java Project

Import the code packages shown in Example 43–1 into the Java source.

Example 43–1 Importing Code Packages to Enable Using the Single Server Task Service API

```
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;
```

43.4.2.3 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 43–2 shows sample code used to declare and obtain task service object references.

Example 43–2 Declaring and Obtaining Task Service Object References

```
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();
```

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.

43.4.2.4 Obtain the Workflow Service Context Object

In order 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 43–3.

Example 43–3 Getting the Workflow Service Context Object

```
// Get the workflow task service context for use in later calls for performance
// improvement
wfCtx = querySvc.getWorkflowContextForAuthenticatedUser();
```

> **Note:** For performance reasons, be sure to pass this context to all subsequent calls to the APIs.

43.4.2.5 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 in order 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 43–4 shows the approval of a task with the ID `0a6d287a-9849-4e5e-914b-805706d6b9d9`.

Example 43–4 Getting the Single Task Object with the Task ID

```
Task t = querySvc.getTaskDetailsById(wfCtxt,
"0a6d287a-9849-4e5e-914b-805706d6b9d9");
taskSvc.updateTaskOutcome(wfCtxt, t, "APPROVE");
// Another example, using the task number to reject a task.
Task t = querySvc.getTaskDetailsByNumber(wfCtxt, 200140 );
taskSvc.updateTaskOutcome(wfCtxt, t, "REJECT");
```

Example 43–5 shows how to use `STDOUT` calls to display the various task attributes through the task API.

Example 43–5 Using STDOUT Calls to Display Task Attributes

```
System.out.println("Task Number: " + task.getSystemAttributes().getTaskNumber());
System.out.println("Task Id: " + task.getSystemAttributes().getTaskId());
System.out.println("Titl    e: " + task.getTitle());
System.out.println("Priority: " + task.getPriority());
System.out.println("State: " + task.getSystemAttributes().getState());
```

43.4.3 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. Once 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.

43.4.3.1 Import Libraries into the Java Project

Import the libraries described in Section 43.4.2.1, "Import Libraries into the Java Project."

43.4.3.2 Import Code Packages into the Java Project

Import the code packages described in Section 43.4.2.2, "Import Code Packages into the Java Project."

43.4.3.3 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 43–6 shows sample code in which task query service object references are declared and obtained.

Example 43–6 Declaring and Obtaining Task Query Service Object References

```
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!";            }
}

IWorkflowServiceClient wfSvcClient =
WorkflowServiceClientFactory.getWorkflowServiceClient(wscct, logger);
taskSvc = wfSvcClient.getTaskService();
querySvc = wfSvcClient.getTaskQueryService();
```

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.

43.4.3.4 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 Section 43.4.5, "How to Query and Traverse Federated and Non-federated Query Result Sets."

43.4.4 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.

43.4.4.1 Import Libraries into the Java Project

Import the libraries described in Section 43.4.2.1, "Import Libraries into the Java Project."

43.4.4.2 Import Code Packages into the Java Project

Import the code packages described in Section 43.4.2.2, "Import Code Packages into the Java Project."

In addition, import the code package shown in Example 43–7.

Example 43–7 Importing the Code Package IFederatedWorkflowContext

```
import oracle.bpel.services.workflow.fws.client.IFederatedWorkflowContext;
```

43.4.4.3 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 43–8 shows sample code in which a list of servers is created for a parallel federated query.

Example 43–8 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");
```

43.4.4.4 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 43–9.

Example 43–9 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);
```

> **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.

43.4.4.5 Obtain the Workflow Service Context

Obtain the workflow service context from the query service, as shown in Example 43–10. This improves performance with all workflow client service API interactions.

Example 43–10 Obtaining the Workflow Service Context

```
fedWFCtx = (IFederatedWorkflowContext) querySvc.getWorkflowContextForAuthenticatedUser();
```

> **Note:** This context is cast as `IFederatedWorkflowContext`. For performance reasons, the context must be passed to all subsequent API calls.

43.4.4.6 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 43–11.

Example 43–11 Implementing Exception Handling

```
// 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("Contextmap: " + fedWFCtx.getWorkflowContextMap());
} // if
```

43.4.4.7 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 Section 43.4.5, "How to Query and Traverse Federated and Non-federated Query Result Sets."

43.4.5 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.
- Programmatically set the task outcome.

43.4.5.1 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 Section 43.4.3, "How to Use the Single Server Task Query Service API" and Section 43.4.4, "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 43–12.

Example 43–12 Sample *.task File Snippet

```
 <?xml version = '1.0' encoding = 'UTF-8'?>
<taskDefinition
targetNamespace="http://xmlns.oracle.com/WFClientPatternSOAApp/WFClientPatternTask
Composite/Humantask1"
xmlns:xp20="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functi
ons.Xpath20"
xmlns:ora="http://schemas.oracle.com/xpath/extension"
```

```
xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.services.functi
ons.ExtFunc"
xmlns:task="http://xmlns.oracle.com/bpel/workflow/task"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://xmlns.oracle.com/bpel/workflow/taskDefinition"
xmlns:evidence="http://xmlns.oracle.com/bpel/workflow/TaskEvidenceService"
xmlns:dvm="http://www.oracle.com/XSL/Transform/java/oracle.tip.dvm.LookupValue"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:ns0="http://xmlns.oracle.com/bpel/workflow/common"
xmlns:hwf="http://xmlns.oracle.com/bpel/workflow/xpath"
xmlns:tsc="http://xmlns.oracle.com/bpel/workflow/common/tsc"
xmlns:xref="http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xpath.XRefXPa
thFunctions"
xmlns:ids="http://xmlns.oracle.com/bpel/services/IdentityService/xpath"
xmlns:mhdr="http://www.oracle.com/XSL/Transform/java/oracle.tip.mediator.service.c
ommon.functions.GetRequestHeaderExtnFunction">
<name>Humantask1</name>
...
```

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 43–13.

Example 43–13 Query task details

```
select * from wftaskmetadata where namespace =
'http://xmlns.oracle.com/WFClientPatternSOAApp/WFClientPatternTaskComposite/Humant
ask1';
```

This query results in the `WFTASKMETADATA` row, shown in Table 43–1 and Table 43–2.

Table 43–1 WFTASKMETADATA Row Part One

ID	URI	Name	Title	Component Name
`default/SOAComposite1!1.0*2008-12-02_08-32-41_642/Humantask1`	`default/SOAComposite1!1.0*2008-12-02_08-32-41_642/Humantask1`	Humantask1	`string('TaskTItle')`	Humantask1

Table 43–2 WFTASKMETADATA Row Part Two

CompositeDN	Composite Name	Composite Version	Namespace
`default/SOAComposite1!1.0*2008-12-02_08-32-41_642`	SOAComposite1	1.0	`http://xmlns.oracle.com/WFClientPatternSOAApp/WFClientPatternTaskComposite/Humantask1 `

You can then query the `WFTASK` table using the task ID shown in the first column of Table 43–1, as shown in Example 43–14.

Example 43–14 Query the WFTASK table using the task ID

```
task:select * from wftask where taskdefinitionid =
'default/SOAComposite1!1.0*2008-12-02_08-32-41_642/Humantask1';
```

The results of the entire table are too large to print, but the selected columns shown in Table 43–3 may be useful.

Table 43–3 Useful Columns

Column Name	Column Value
State	`ASSIGNED`
TaskID	`0a6d287a-9849-4e5e-914b-805706d6b9d9`
TaskNumber	`200140`
WorkflowDescriptorURI	`default/SOAComposite1!1.0*2008-12-02_08-32-41_642/Humantask1`
TaskDefinitionID	`default/SOAComposite1!1.0*2008-12-02_08-32-41_642/Humantask1`
TaskDefinitionName	`Humantask1`
CorrelationID	`0a6d287a-9849-4e5e-914b-805706d6b9d9`

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`.

43.4.5.2 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 43–15.

Example 43–15 Constructing the Predicate

```
new Predicate(TableConstants.WFTASK_TASKNUMBER_COLUMN, Predicate.OP_EQ, 200140);
```

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 43–16.

Example 43–16 Creating Namespace, State and Combination Predicates

```
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);
```

An additional way to construct a predicate is shown in Example 43–17, as specified in the `ITaskQueryService` documentation.

Example 43–17 Another Way to Construct a Predicate

```
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);
```

43.4.5.3 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 43–18, the `TITLE_COLUMN` and `PRIORITY_COLUMN` properties are added to the `ORDER_BY` clause.

Example 43–18 Constructing the Ordering of queryTasks()

```
// Create the ordering
Ordering ordering = new Ordering(TableConstants.WFTASK_TITLE_COLUMN, true, true);
ordering.addClause(TableConstants.WFTASK_PRIORITY_COLUMN, true, true);
```

43.4.5.4 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 43–19 shows sample code that constructs a list of display columns for `queryTasks()`.

Example 43–19 Constructing the List of Display Columns for queryTasks()

```
// List of display columns
// For those columns that are not specified here, the queried Task object will not
// hold any value.
// For example: If TITLE is not specified, task.getTitle() returns a value of
// null.
// For the list of most comonly used columns, check the table below
// Note: TASKID is fetched by default, such that it is unnecessary to explicitly
// specify it.
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");
```

43.4.5.5 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 43–20.

Example 43–20 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 permoded by the user
// as a member of a group).
//                - task.getSystemAttributes().getSystemActions()
// "ShortHistory" - task.getSystemAttributes().getShortHistory()
List optionalInfo = new ArrayList();
optionalInfo.add("Actions");
```

43.4.5.6 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 43–21.

Example 43–21 Query Service API

```
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)
```

The `queryTasks()` method returns a list of tasks that match the predicate and ordering criterion.

Example 43–22 shows how to invoke `queryTasks()` using the previously constructed attribute lists.

Example 43–22 Invoking queryTasks() with the Previously Constructed Attribute Lists

```
List tasksList = querySvc.queryTasks(wfCtxt,
                                     queryColumns,
                                     optionalInfo,
                                     ITaskQueryService.ASSIGNMENT_FILTER_MY_AND_GROUP,
                                     keyword,
                                     predicate,
                                     ordering,
                                     0,0); // No Paging
```

More information on paging is available in the API documentation. (Look for the text "How to use paging.")

43.4.5.7 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 43–23 shows how to iterate through the result set.

Example 43–23 Iterating through the Result Set

```
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.getSystemAttributes().getTaskId());
     System.out.println("Title: " + task.getTitle());
     System.out.println("Priority: " + task.getPriority());
     System.out.println("State: " + task.getSystemAttributes().getState());
     System.out.println();
     // Retrive any Optional Info specified
     // Use task service, to perform operations on the task
     str = str + ":";   }
}
```

43.4.5.8 Programmatically Set the Task Outcome

Once 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 43–24
- Example 43–25
- Example 43–26
- Example 43–27
- Example 43–28
- Example 43–29

Example 43–24 Programmatically Setting the Task Outcome

```
updateTaskOutcome(IWorkflowContext context,Task task, java.lang.String outcome)
// Set the outcome of the task.
```

Example 43–25 Setting the Outcome of the Task Reference on the Single Server Task Service

```
'APPROVE'.taskSvc.updateTaskOutcome(wfCtxt, t, "APPROVE");
```

Example 43–26 Setting the Outcome of the Task Reference on the Single Server Task Service

```
'REJECT'.taskSvc.updateTaskOutcome(wfCtxt, t, "REJECT");
```

Example 43–27 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 43–28 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 43–29 Updating a Single Task Outcome on the Federated Query Service

```
Map<String, IWorklowContext> ctxMap = fedWFCtx.getWorkflowContextMap()
String serverName = task.getServerName();
IWorklowContext taskWfCtx = ctxMap.get(serverName);

if (taskWfCtx !=null)
```

43.5 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.

43.6 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 Chapter 50, "Securing Web Services Use Cases."

43.7 Verifying the Deployment

Validating your implementation involves deploying human tasks to a SOA domain, in the following steps.

- Section 43.7.1, "Deploying the Human Task"
- Section 43.7.2, "Deploying Programmatic Task Functionality"
- Section 43.7.3, "Invoking Programmatic Task Functionality"

43.7.1 Deploying 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. Once the task is deployed and registered, your code or BPEL can initiate the task and facilitate task resolution through the code or worklist. For more information, see "Part V: Using the Human Workflow Service Component" in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

43.7.2 Deploying 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. For more information, see "Deploying SOA Composite Applications" in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

43.7.3 Invoking 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.

`http://host:port/integration/worklistapp`

43.8 Troubleshooting the Use Case

Following are some suggestions for troubleshooting task data and the Java code in the use case.

43.8.1 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.

43.8.2 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.

43.9 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.

44

Implementing an Oracle ADF Task Flow for a Human Task

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

44.1 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.

44.2 Other Approaches

There are no other supported approaches to this use case.

44.3 Example

The sample code for this use case can be downloaded from Oracle SOA Suite samples.

44.4 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.

44.4.1 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:

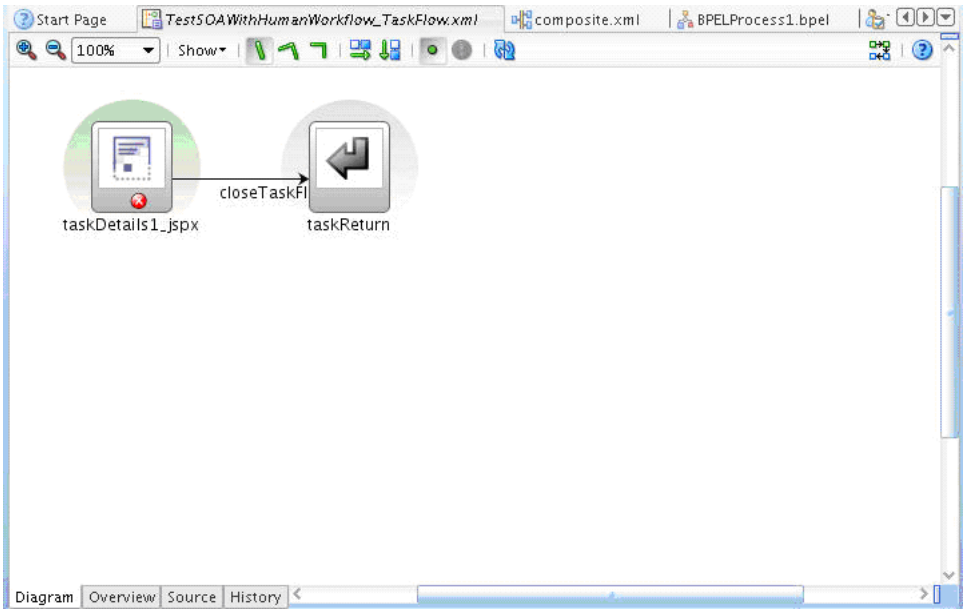
1. Right-click your UI project and choose **New**.
2. In the New Gallery window, select **Web Tier > JSF > ADF Task Flow Based on Human Task** and click **OK**.
3. In the SOA Resource Browser dialog, select the TASK file location and target TASK file.

   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 44–1. Rename the view activity to something meaningful to you.

Figure 44–1 Task Flow with View Component



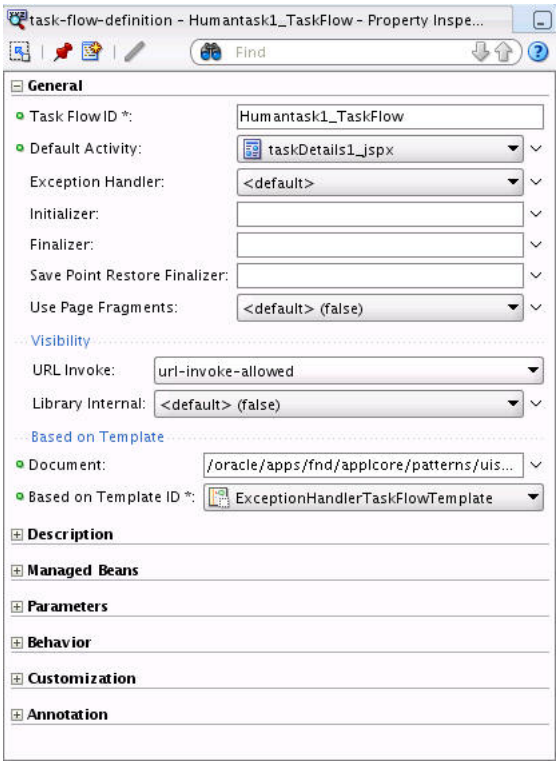
5. Double-click the view activity in the task flow. From the Create JSF Page dialog that displays, modify the file name or directory location as needed, and click **OK**.

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 44–2.

Figure 44–2 Add the template with the Property Inspector



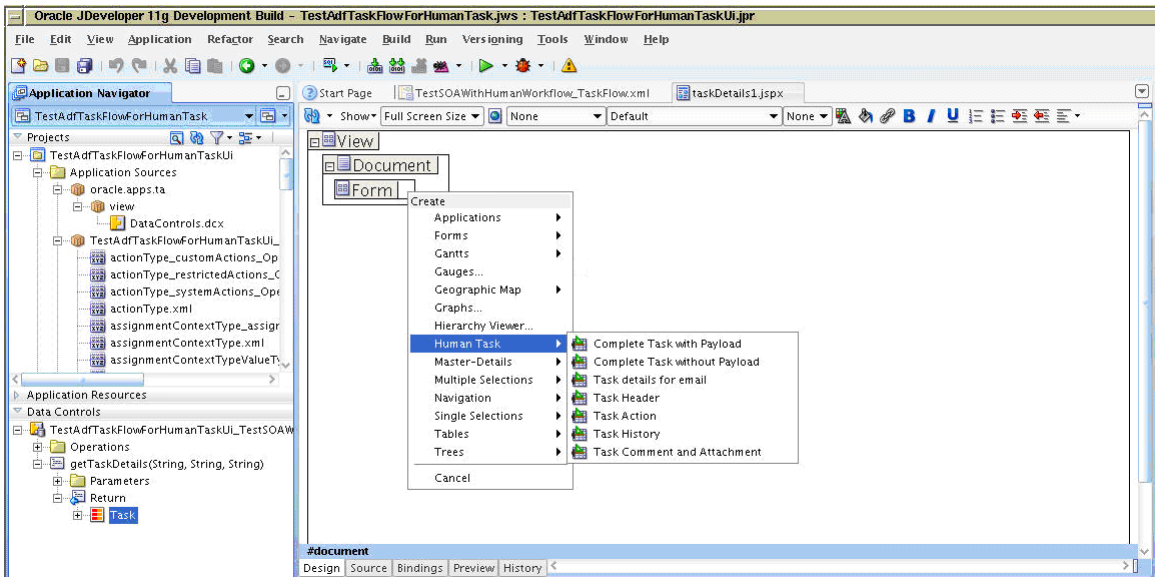
44.4.2 Creating a User Interface for the Human Task

Use the drop handler template to create a user interface for the human task.

To create a user interface:

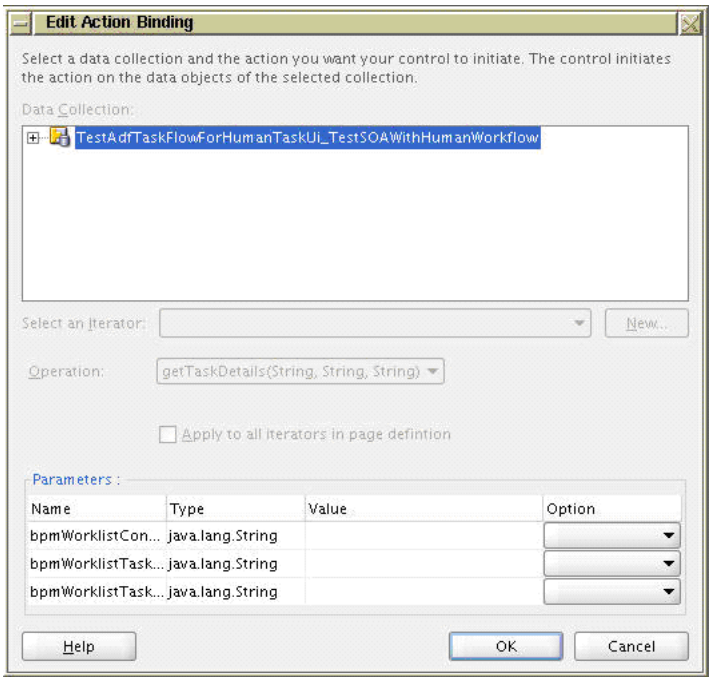
1. Create a task detail UI and navigate to **Application Navigator > Data Controls >** ***Task Data Control Name*** **> getTaskDetails > Return > Task**.
2. Drag and drop the task data control to your JSPX page and select **Create > Human Task > Complete Task with Payload**, as shown in Figure 44–3.

Figure 44–3 Complete Task with Payload



The Edit Action Binding (`getTaskDetails`) dialog displays, as shown in Figure 44–4.

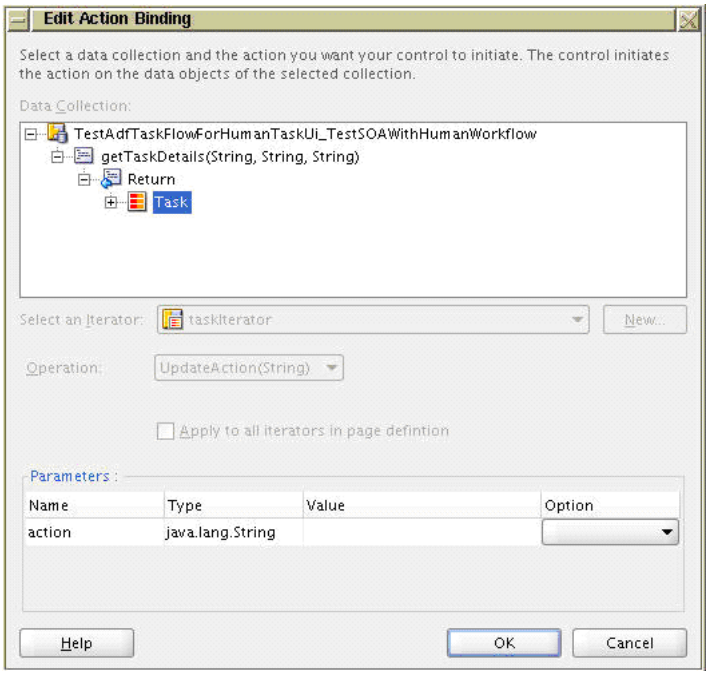
Figure 44–4 Edit Action Binding (getTaskDetails) Dialog



3. In the Edit Action Binding (`getTaskDetails`) dialog, click **OK**.

   The Edit Action Binding (`UpdateActions`) dialog displays, as shown in Figure 44–5.

Figure 44–5 Edit Action Binding (UpdateActions) Dialog



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 44–6 shows the Source view of a sample JSPX page.

Figure 44–6 <af:form> Component

```
<?xml version='1.0' encoding='US-ASCII'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
          xmlns:f="http://java.sun.com/jsf/core"
          xmlns:h="http://java.sun.com/jsf/html"
          xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
          xmlns:trh="http://myfaces.apache.org/trinidad/html"
          xmlns:wf="http://xmlns.oracle.com/bpel/workflow/workflow-taglib.tld"
          xmlns:wlc="http://xmlns.oracle.com/bpel/workflow/worklist"
          xmlns:fnd="http://xmlns.oracle.com/apps/fnd/applcore">
  <jsp:directive.page contentType="text/html;charset=US-ASCII"/>
  <f:view locale="#{applCorePrefs.locale}">
    <af:document id="d1" title="#{bindings.title.inputValue}">
      <af:messages id="m1"/>
      <f:loadBundle basename="oracle.bpel.services.workflow.worklist.resources.worklist"
                    var="resources"/>
      <trh:script source="/jsLibs/taskDetails.js"></trh:script>
      <af:form id="f1" usesUpload="true">
        <af:panelHeader text="#{bindings.title.inputValue}">
          <f:facet name="context"/>
          <f:facet name="menuBar"/>
          <f:facet name="toolbar">
            <af:toolbar visible="#{actionAvailable.toolbarAvailable}" id="t3">
              <af:group rendered="#{!(actionAvailable.resumeAvailable || actionAvailable.claimAvailable || actionAvailable.is
                        id="g1">
                <af:menuBar id="mb1">
                  <af:menu id="acts" text="#{resources.TASK_ACTIONS}"
                           visible="#{actionAvailable.actionsAvailable || bindings.update.enabled}">
                    <af:group id="g2">
                      <af:forEach var="childNode1"
                                  items="#{actionAvailable.customActions}">
                        <af:commandMenuItem textAndAccessKey="#{childNode1.displayName}"
                                            actionListener="#{invokeActionBean.setAction}"
                                            action="#{invokeActionBean.action}"
                                            partialSubmit="false" id="cmi2">
                          <f:attribute name="ACTION_MENU_ITEM"
                                       value="#{childNode1.action}"/>
```

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).

44.4.3 Implementing Product-Specific Sections

Product-specific sections include the following:

- Instructions
- Details
- Recommended Actions
- <PLACE APPLICATION SPECIFIC CONTENT HERE>
- Related Links
- Comments and Attachments
- History

> **Note:** Do not modify anything in the portion of the template that includes the following:
>
> - Title attribute of the of the `<af:document>` component
> - Text attribute of the `<af:panelHeader>` component
> - Toolbar facet of the `<af:panelHeader>` component

44.4.3.1 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 44–1.

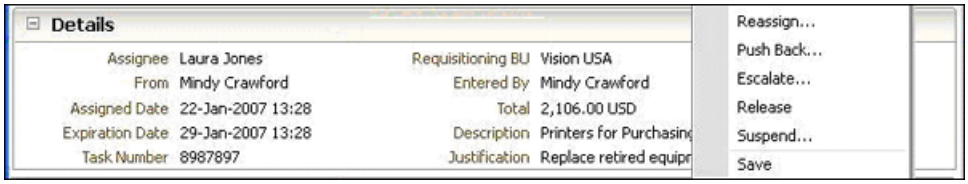
Example 44–1 Adding instructions above the Details section

```
<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">
```

44.4.3.2 How to Modify Details

The Details section, as shown in Figure 44–7, 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 44–7 Details Section



Do not modify the code in the first column, which corresponds to the first `<trh:cellFormat>` with `id="cf1"`, as shown in Figure 44–7.

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 44–2.

Example 44–2 Modify the third <trh:cellFormat> component

```
<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>
```

> **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 an additional column for spacing, `<trh:cellFormat id="cf7">`. Add an additional 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 44–3.

Example 44–3 Adding product-specific information

```
<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">
            ... Third column containing product-specific fields ...
         </trh:cellFormat>
       </trh:rowLayout>
   </trh:tableLayout>
</af:showDetailHeader>
```

44.4.3.3 How to Modify Recommended Actions

The Recommended Actions section, as shown in Figure 44–8, is used in the Information Only pattern.

Figure 44–8 Recommended Actions Section

The Oracle ADF code for the Recommended Actions `<af:showDetailHeader>` component is shown in Example 44–4.

Example 44–4 Code for Recommended Actions

```
<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>
```

- 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.

44.4.3.4 How to Modify <PLACE APPLICATION SPECIFIC CONTENT HERE>

The <PLACE APPLICATION SPECIFIC CONTENT HERE> section is a place holder 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> `<af:showDetailHeader>` component is shown in Example 44–5.

Example 44–5 Oracle ADF code for the application specific content section

```
<af:showDetailHeader size="1" id="applicationContentHeader" text="&lt;PLACE
APPLICATION SPECIFIC CONTENT HERE&gt;" 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}"
required="#{bindings.PayloadInput1.hints.mandatory}"
columns="#{bindings.PayloadInput1.hints.displayWidth}"
maximumLength="#{bindings.PayloadInput1.hints.precision}"
shortDesc="#{bindings.PayloadInput1.hints.tooltip}" id="it3">
                <f:validator binding="#{bindings.PayloadInput1.validator}"/>
           </af:inputText>
      </af:panelFormLayout>
   </af:panelGroupLayout>
</af:showDetailHeader>
```

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 Section 44.4.3.5, "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.

44.4.3.5 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 44–6.

Example 44–6 Code that Results from Implementing Links

```
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)
```

44.4.3.6 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 44–7.

Example 44–7 Modifying comments and attachments

```
<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>
```

- 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 44–8.

Example 44–8 Adding additional instructions to the Attachments dialog

```
<af:popup id="popupAddAttachmentDialog">
   <af:dialog title="#{resources.ADD_ATTACHMENT}" okVisible="false"
              cancelVisible="false" closeIconVisible="false" id="d2">
      ...
      <af:panelGroupLayout id="pgl10">
         <f:facet name="separator">
            <af:spacer width="15" height="15" id="s2"/>
         </f:facet>
         <af:outputText value="#{resources.UPLOAD_FILE_CAVEAT}" id="ot10"/>
         ... Add additional instructions here ...
         <af:panelFormLayout id="pfl2">
            <af:selectOneRadio label="#{resources.ATTACH_TYPE}"
              value="#{readAttachmentBean.selectedAttachmentType}"
              valueChangeListener="#{readAttachmentBean.toggle}" autoSubmit="true"
              id="editAttachmentType" layout="horizontal" immediate="true">
      ...
   </af:afdialog>
</af:popup>
```

44.4.3.7 How to Modify Related Links

The Related Links section is used in the Information Only pattern.

Example 44–9 shows the Oracle ADF code for the Related Links `<af:showDetailHeader>` component.

Example 44–9 Modifying Related Links

```
<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"/>
   <f:facet name="context"/>
</af:showDetailHeader>
```

- 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 Section 44.4.3.5, "How to Implement Links."

44.4.3.8 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 44–10.

Example 44–10 Modifying task history

```
<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>
```

44.4.4 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:

- Section 44.4.1, "Creating an Oracle ADF Task Flow"
- Section 44.4.2, "Creating a User Interface for the Human Task"
- Section 44.4.3, "Implementing Product-Specific Sections"
- Section 44.4.4, "Implementing a Task Detail with Contextual Area"

44.4.5 Implementing 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.

44.4.5.1 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 44–9.

Figure 44–9 Notification Settings

44.4.5.2 Determining 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
  - af:selectOneChoice
  - af:showDetailHeader (excludes `helpTopicId` attribute for instructions)
  - af:table
  - trh:tableLayout
  - trh:rowLayout
  - trh:cellFormat
  - af:panelFormLayout

  - af:panelGroupLayout
  - af:panelList
  - af:spacer
- 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 a switcher component in the JSPX page to ensure that only supported components are rendered in the email version. For more information, see Section 44.4.5.3, "Using a Switcher Component."

44.4.5.3 Using a Switcher Component

Introduce a switcher component in your JSPX page when you have components that are not supported in the email notification.

To use a switcher 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 44–11.

Example 44–11 Using a switcher component

```
<af:switcher 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:switcher>
```

3. If it is not a `notificationClient`, the value of `bpmClientType` is null. Continue to Section 44.4.5.5, "Fine-Tuning the Emailable Page."

44.4.5.4 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 44–12.

Example 44–12 Adding two cases to the switcher

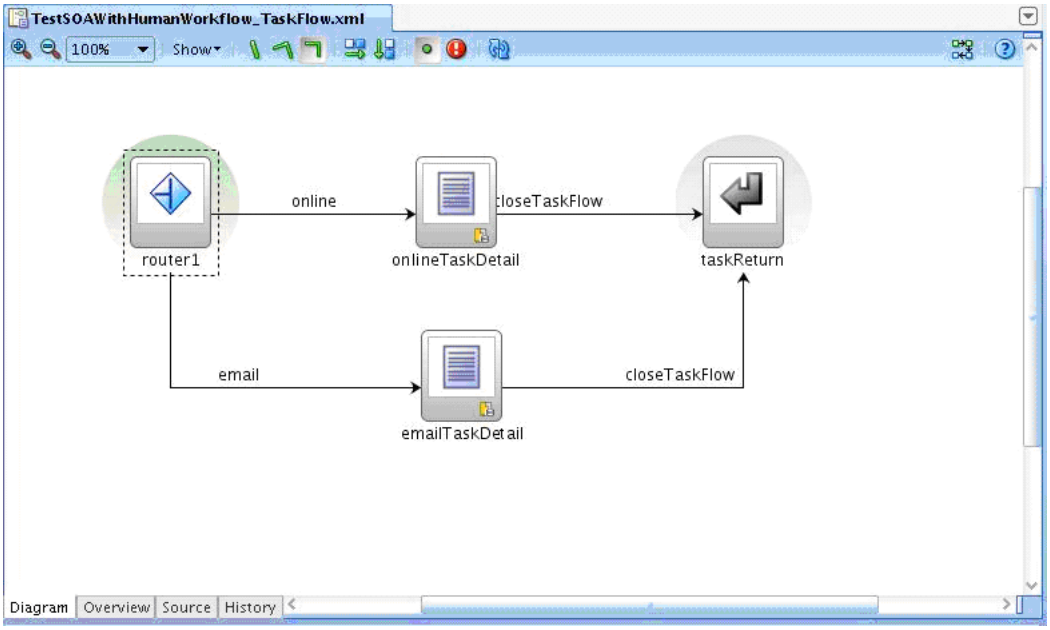
```
Expression = "#{pageFlowScope.bpmClientType=="notificationClient"}" and Outcome =
"email"

Expression = "#{empty pageFlowScope.bpmClientType}" and Outcome = "online"
```

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 44–10.

Figure 44–10 Setting Default Activity



44.4.5.5 Fine-Tuning the Emailable Page

In order 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 44–13.

Example 44–13 Adding an EL expression to the disclosed property

```
<af:showDetailHeader size="1"
                     id="histHd"
                     text="#{resources.HISTORY}"
                     disclosed="#{pageFlowScope.bpmClientType ==
'notificationClient'}"
...
```

- 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.

44.4.6 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. For more information, see the chapter "Internationalizing and Localizing Pages" in the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

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 44–14, replace the `"#{bindings.PayloadInput1.hints.label}"` so that it refers to translated text in the Xliff resource bundle associated with your UI project.

Example 44–14 Label attributes for the payload

```
<af:inputText value="#{bindings.PayloadInput1.inputValue}"
              label="#{bindings.PayloadInput1.hints.label}"
              required="#{bindings.PayloadInput1.hints.mandatory}"
              columns="#{bindings.PayloadInput1.hints.displayWidth}"
              maximumLength="#{bindings.PayloadInput1.hints.precision}"
              shortDesc="#{bindings.PayloadInput1.hints.tooltip}"
              id="it1">
  <f:validator binding="#{bindings.PayloadInput1.validator}"/>
</af:inputText>
```

> **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 Chapter 61, "Creating Customizable Applications."

44.4.7 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.

44.4.8 Configuring 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 Chapter 3, "Setting Up Your JDeveloper 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 44–15.

Example 44–15 Adding the Oracle ADF library to the SuperWeb project

```
<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.SecureNotificationS
    ervlet
</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>
```

3. Under the default source location, create the `hwtaskflow.xml` file and merge all the notification task flow details.

44.5 Securing the Design Pattern

To secure this pattern, follow the instructions described in Chapter 50, "Securing Web Services Use Cases."

You may want to implement the following security tasks:

- Secure task flows (bounded)
- Secure page fragments
- Secure actions

44.6 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. 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 *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

If you want to deploy the Oracle ADF task flow for human task to a SOA server, deploy your application as described in the chapter "Deploying SOA Composite Applications" in *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

To verify the deployment:

1. Deploy the `oracle.soa.workflow` shared library to 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 WebLogic server.
   b. Select **Deployments** and click **Install**.
   c. In the **Path** field, make sure the following value is specified.

      ```
      <JDEV_HOME>/fmwtools/fmwtools_home/jdeveloper/
      soa/modules/oracle.soa.workflow_11.1.1
      ```

   d. Select `oracle.soa.workflow.jar` and click **Finish**.
   e. 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.

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 WebLogic server.

b. Navigate to **Domain Structure > Services > Foreign JNDI Providers** and click **New**.

c. Enter the name `ForeignJNDIProvider-SOA`, and click **OK**.

d. Click **ForeignJNDIProvider-SOA**.

e. Press **Enter** and then click **Save**.

f. 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 c and 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 44–16.

Example 44–16 Grant for bpm-services.jar

```
<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>
```

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.

44.7 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.

44.7.1 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 44–17 shows a snippet of a sample `weblogic-application.xml` file with the correct reference to `oracle.soa.workflow.wc`.

Example 44–17 Reference oracle.soa.workflow.wc

```
<?xml version = '1.0' encoding = 'US-ASCII'?>
<weblogic-application xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.bea.com/ns/weblogic/weblogic-
application http://www.bea.com/ns/weblogic/weblogic-
application/1.0/weblogic-application.xsd"
xmlns="http://www.bea.com/ns/weblogic/weblogic-application">

...
  <library-ref>
    <library-name>oracle.soa.workflow.wc</library-name>
  </library-ref>
...

</weblogic-application>
```

44.7.2 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 44–18.

Example 44–18 Set the FRAME_BUSTING attribute in web.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.

44.7.3 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.

44.7.4 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 44–19 shows a sample illustrating `<af:commandImageLink>` after modifying the action property value.

Example 44–19 Modifying the action property value for <af:commandImageLink>

```
<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"/>
```

45

Cross Family Business Event Subscription Pattern

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 aqueue 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

45.1 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.

45.2 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.

45.3 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 45–1 lists the HCM composite components and values.

Table 45–1 HCM Composite Components and Values

Composite Components	Values
Publishing Product	`Per`
Subscribing Product	`Hz`
Owning Product	`Hz`
XFamilyPub composite name	`FoundationPartiesPerToHzXFamilyPubComposite`
XFamilySub composite name	`FoundationPartiesHzXFamilySubComposite`
Recipient/Consumer	`FoundationPartiesPerToHz`
SOA workspace	`CrmCommonSoa.jws`

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 45–2 lists the CRM composite components and values.

Table 45–2 CRM Composite Components and Values

Composite Components	Values
Publishing Product	`Hz`
Subscribing Product	`Per`

Table 45–2 (Cont.) CRM Composite Components and Values

Composite Components	Values
Owning Product	`Hz`
XFamilyPub composite name	`FoundationPartiesHzToPerXFamilyPubComposite`
XFamilySub composite name	`FoundationPartiesPerXFamilySubComposite`
Recipient/Consumer	`FoundationPartiesHzToPer`
SOA workspace	`CrmCommonSoa.jws`

The sample code for this use case can be downloaded from Oracle SOA Suite samples.

45.4 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.

45.4.1 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.
- 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 Resources > Deployments > AqAdapter > Configuration > Outbound Connection Pools**. The connection pool JDBC/XFamilyEventAqueue should display.

45.4.2 Determining 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.

45.4.3 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

45.4.4 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 Section 45.4.2.
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`.
   c. In the Interface field, define the operation and schema.
   d. For the operation type, select the **Enqueue** radio button. In the Operation Name field, enter **Enqueue**.
   e. For the queue name, enter **FUSION** as the database schema and **ACR_XFAMILY_EVENT_Q** for the queue name.
   f. On the Queue Parameters page, enter the recipient as defined in Section 45.4.3.
   g. 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. Once you complete this step, the error icon that displayed in the previous step disappears.

Example 45–1 shows a sample XSL file.

Example 45–1 Sample XSL File

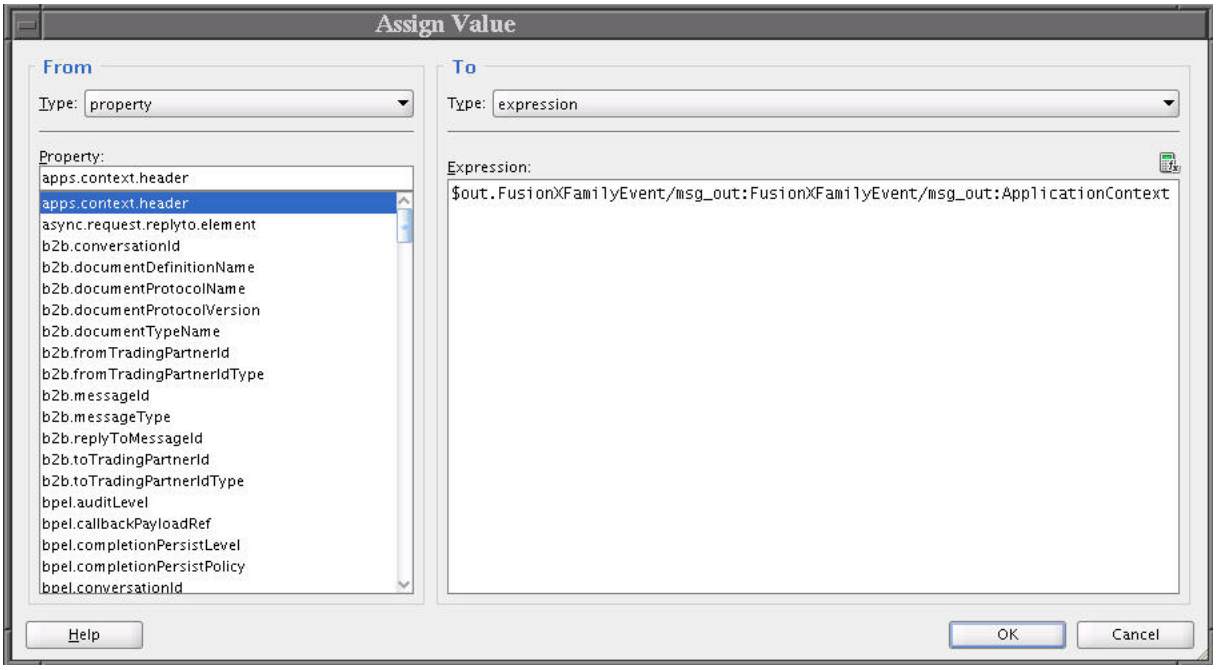
```
<?xml version="1.0" encoding="UTF-8" ?>
<?oracle-xsl-mapper
  <!-- SPECIFICATION OF MAP SOURCES AND TARGETS, DO NOT MODIFY. -->
  <mapSources>
    <source type="XSD">
      <schema location="../xsd/bulkImportEvents.xsd"/>
      <rootElement name="batchInfo"namespace="http://xmlns.oracle.com/apps/crm/hz/bulkImport/
        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]. -->
?>
<xsl:stylesheet version="1.0"
                xmlns:xpath20="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.
                services.functions.Xpath20"
                xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
                xmlns:mhdr="http://www.oracle.com/XSL/Transform/java/oracle.tip.mediator.
                service.common.functions.MediatorExtnFunction"
                xmlns:oraext="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.
                              services.functions.ExtFunc"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:dvm="http://www.oracle.com/XSL/Transform/java/oracle.tip.dvm.LookupValue"
                xmlns:hwf="http://xmlns.oracle.com/bpel/workflow/xpath"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:med="http://schemas.oracle.com/mediator/xpath"
                xmlns:ids="http://xmlns.oracle.com/bpel/services/IdentityService/xpath"
                xmlns:xdk="http://schemas.oracle.com/bpel/extension/xpath/function/xdk"
                xmlns:xref="http://www.oracle.com/XSL/Transform/java/oracle.
                            tip.xref.xpath.XRefXPathFunctions"
                xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                xmlns:ns0="http://xmlns.oracle.com/apps/crm/hz/bulkImport/
                           FoundationBulkImportEventsComposite"
                xmlns:ora="http://schemas.oracle.com/xpath/extension"
                xmlns:socket="http://www.oracle.com/XSL/Transform/java/oracle.
                              tip.adapter.socket.ProtocolTranslator"
                xmlns:msg_out="http://xmlns.oracle.com/apps/common/acr/events"
                xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/XFamilyEventPattern/
                FinInfrHzToZxXFamilyPubComposite/EnqueueEventMessage"
                xmlns:ldap="http://schemas.oracle.com/xpath/extension/ldap"
                exclude-result-prefixes="xsi xsl xsd ns0 plt wsdl msg_out tns xpath20 bpws mhdr
                oraext dvm hwf med ids xdk xref ora socket ldap">
  <xsl:template match="/">
    <msg_out:FusionXFamilyEvent>
      <msg_out:Namespace>
        <xsl:text disable-output-escaping="no">http://xmlns.oracle.com/apps/crm/hz/bulkImport/
          FoundationBulkImportEventsComposite</xsl:text>
      </msg_out:Namespace>
      <msg_out:LocalName>
        <xsl:text disable-output-escaping="no">ImportPartyData</xsl:text>
      </msg_out:LocalName>
      <msg_out:Payload>
        <xsl:copy-of select="/ns0:batchInfo"/>
      </msg_out:Payload>
    </msg_out:FusionXFamilyEvent>
  </xsl:template>
</xsl:stylesheet>
```

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 45–1.

   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 45–1 Define an Assign Statement



The mediator source code is shown in Example 45–2.

Example 45–2 Mediator Source Code

```
<assign>
   <copy target="$out.FusionXFamilyEvent/msg_out:FusionXFamilyEvent/msg_out:ApplicationContext"
         value="$in.property.apps.context.header"
         xmlns:msg_out="http://xmlns.oracle.com/apps/common/acr/events"/>
</assign>
```

6. Open the composite.xml file and search for the following line.

   ```
   <binding.jca config="EnqueueEventMessage_aq.jca"/>
   ```

7. Replace the `binding.jca` element with the code shown in Example 45–3.

Example 45–3 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">xmlns:msg_out=
                 "http://xmlns.oracle.com/apps/common/acr/events"</property>
</binding.jca>
```

8. Add standard fault policies to the composite just above the `<component>` elements in the composite.xml file, as shown in Example 45–4.

Example 45–4 Add Fault Policies to the composite.xml File

```
<property
name="oracle.composite.faultPolicyFile">oramds:/apps/oracle/apps/fnd/applcore/soa/fault/fault-polic
ies.xml</property>
<property
name="oracle.composite.faultBindingFile">oramds:/apps/oracle/apps/fnd/applcore/soa/fault/fault-bind
ings.xml</property>
```

45.4.5 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 Section 45.4.2, "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.

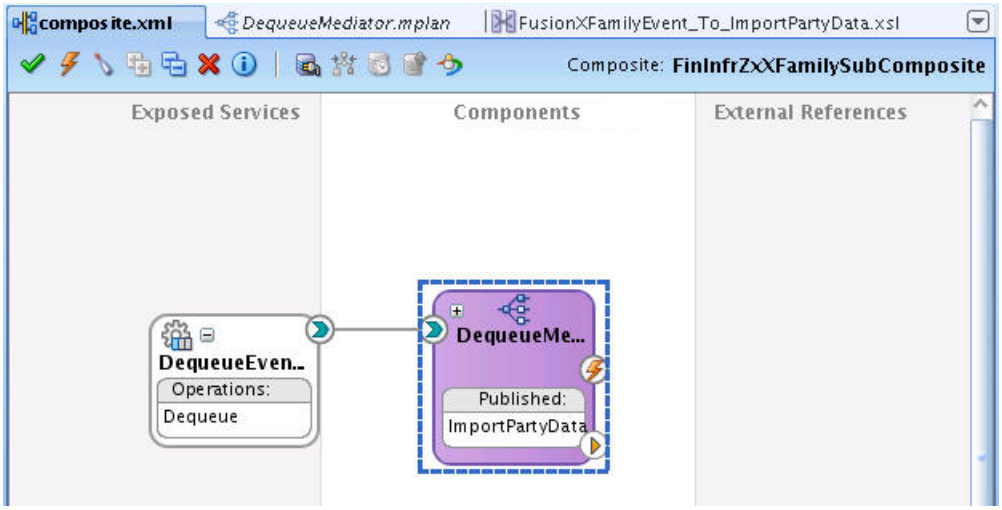
      In the Queue Name field, enter **ACR_XFAMILY_EVENT_Q**.
   f. In the Queue Parameters window, specify the aqueue message recipient as defined in Section 45.4.3, "Determining the Aqueue Message Recipient."
   g. In the Messages window, select the message schema URL and the relevant schema element.

      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 45–2.

Figure 45–2 Connect the Mediator to the Aqueue Adaptor



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 45–5.

Example 45–5 A Filter Expression for the Mediator Rule

```
(($in.FusionXFamilyEvent/msg_out:FusionXFamilyEvent/msg_out:Namespace =
"http://xmlns.oracle.com/apps/crm/hz/bulkImport/FoundationBulkImportEventsComposite")
and ($in.FusionXFamilyEvent/msg_out:FusionXFamilyEvent/msg_out:LocalName = "ImportPartyData"))
```

   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 45–6.

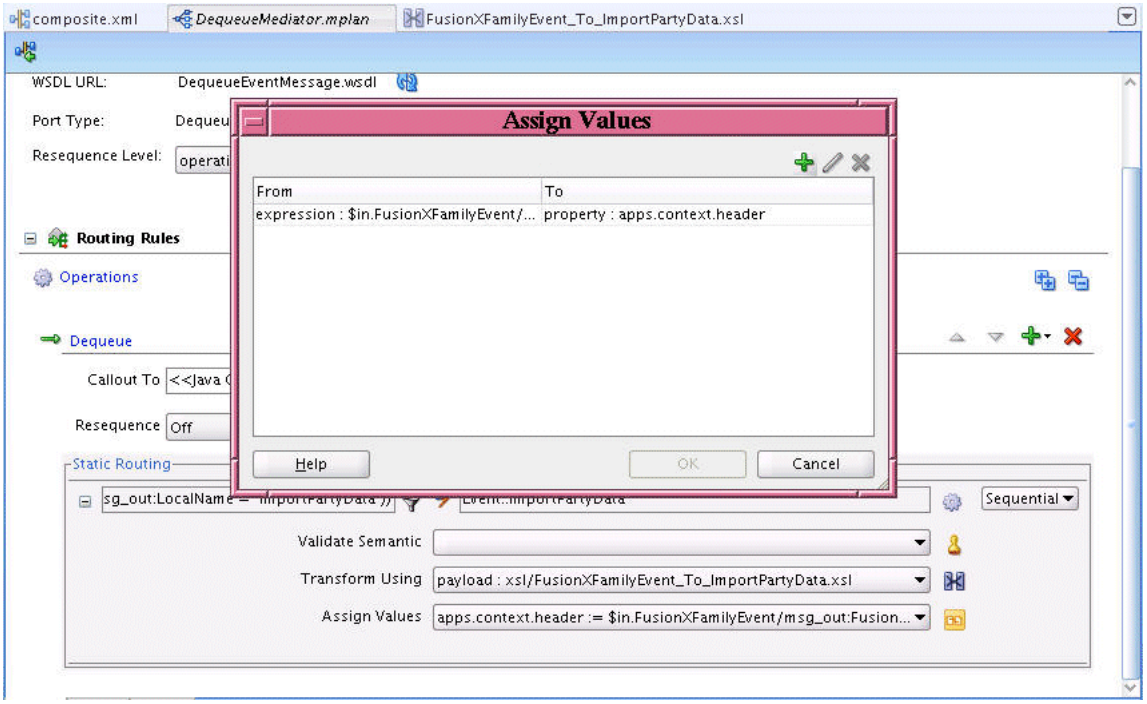
Example 45–6 XSL File

```
<?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]. -->
?>
<xsl:stylesheet version="1.0"
                xmlns:xpath20="http://www.oracle.com/XSL/Transform/java/oracle.
                               tip.pc.services.functions.Xpath20"
                xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
                xmlns:mhdr="http://www.oracle.com/XSL/Transform/java/oracle.tip.mediator.
                service.common.functions.MediatorExtnFunction"
                xmlns:oraext="http://www.oracle.com/XSL/Transform/java/oracle.
                              tip.pc.services.functions.ExtFunc"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:dvm="http://www.oracle.com/XSL/Transform/java/oracle.tip.dvm.LookupValue"
                xmlns:hwf="http://xmlns.oracle.com/bpel/workflow/xpath"
                xmlns:tns="http://xmlns.oracle.com/pcbpel/adapter/aq/XFamilyEventPattern/
                           FinInfrZxXFamilySubComposite/DequeueEventMessage"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:med="http://schemas.oracle.com/mediator/xpath"
                xmlns:ids="http://xmlns.oracle.com/bpel/services/IdentityService/xpath"
                xmlns:xdk="http://schemas.oracle.com/bpel/extension/xpath/function/xdk"
                xmlns:xref="http://www.oracle.com/XSL/Transform/java/oracle.
                            tip.xref.xpath.XRefXPathFunctions"
                xmlns:plt="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                xmlns:ns0="http://xmlns.oracle.com/apps/crm/hz/bulkImport/
                           FoundationBulkImportEventsComposite"
                xmlns:ora="http://schemas.oracle.com/xpath/extension"
                xmlns:socket="http://www.oracle.com/XSL/Transform/java/oracle.tip.adapter.socket.
                              ProtocolTranslator"
                xmlns:msg_out="http://xmlns.oracle.com/apps/common/acr/events"
                xmlns:ldap="http://schemas.oracle.com/xpath/extension/ldap"
                exclude-result-prefixes="xsi xsl tns plt xsd wsdl msg_out ns0 xpath20 bpws
                              mhdr oraext dvm hwf med ids xdk xref ora socket ldap">
  <xsl:template match="/">
    <xsl:copy-of select="/msg_out:FusionXFamilyEvent/msg_out:Payload/child::node()">
      <?oracle-xsl-mapper-position ns0:batchInfo?>
    </xsl:copy-of>
  </xsl:template>
</xsl:stylesheet>
```

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 45–3.

Figure 45–3 Defining an Assign Statement to Copy the Context



5. In the composite.xml file, replace the line `<binding.jca config="EnqueueEventMessage_aq.jca"/>` with the code shown in Example 45–7.

Example 45–7 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>
```

6. Add standard fault policies to the composite, as shown in Example 45–8.

Example 45–8 Adding Standard Fault Policies

```
<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>
```

45.5 Verifying 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.

- Section 45.5.1, "How to Verify the Deployment of the XFamilyPub Composite"

- Section 45.5.2, "How to Verify the Deployment of the XFamilySub Composite"

45.5.1 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 45–9 shows sample PL/SQL code used to raise a business event.

Example 45–9 Raise a Business Event by Invoking the PL/SQL Code

```
fnd_global.initialize_session('F58679122D280FD03AD1198A55EFC6BF', 'Abraham.Mason');

hcm_edn_publish_event('/oracle/apps/hcm/users/publicModel/entity/events/edl/LdapRequestEO',
     'ActiveLdapRequestEvent',

         '<business-event xmlns="http://oracle.com/fabric/businessEvent"
               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">'||

         '<LdapRequestId>'||

            '<newValue value="5"/>'||

            '<oldValue value="5"/>'||

         '</LdapRequestId>'||

      '</ActiveLdapRequestEventInfo>'||

                    '</content>'||

                    '</business-event>');

commit;

end;

/
```

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:

    ```
    select user_data from fusion.acr_xfamily_event_qt;
    ```

45.5.2 How to Verify the Deployment of the XFamilySub Composite

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.

Part VII

Implementing Security

This part of the Developer's Guide provides information about Oracle Fusion Applications security. It discusses how to implement Oracle Fusion Data Security and user sessions, and how to secure specific use cases for Oracle ADF application artifacts, Web services, and portlet applications.

Getting Started with Security introduces security concepts and features, 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.

Implementing Oracle Fusion Data Security describes how to enforce authorization for access and modification of specific data records. The goal of Oracle Fusion Data Security is to authorize a user to perform specified actions on selected data. Data security can secure rows and attributes of a database object and addresses the question "Who can do what on which set of data."

Implementing Application User Sessions describes how to allow applications to store security and application context on the user session, and to allow for an enhanced security implementation. An 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. User session roles can be enabled for a user, and dictate what privileges that user has.

Implementing Function Security describes how to authorize end users to access securable application artifacts created using Oracle ADF.

Securing Web Services Use Cases describes best practices for for securing Web services in Oracle Fusion Applications and specifically explains the difference between global policy attachment and local policy attachment and when to use each.

Securing End-to-End Portlet Applications describes how to authenticate and authorize portlet services, as well as how to configure key stores and credential stores.

This part contains the following chapters:

46

Getting Started with Security

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:

46.1 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 polices are properly enforced throughout the application. The implementation details may vary depending on the technology used in the application.

For complete details about the Oracle Fusion security approach, including concepts and best practices, see the *Oracle Fusion Applications Security Guide*.

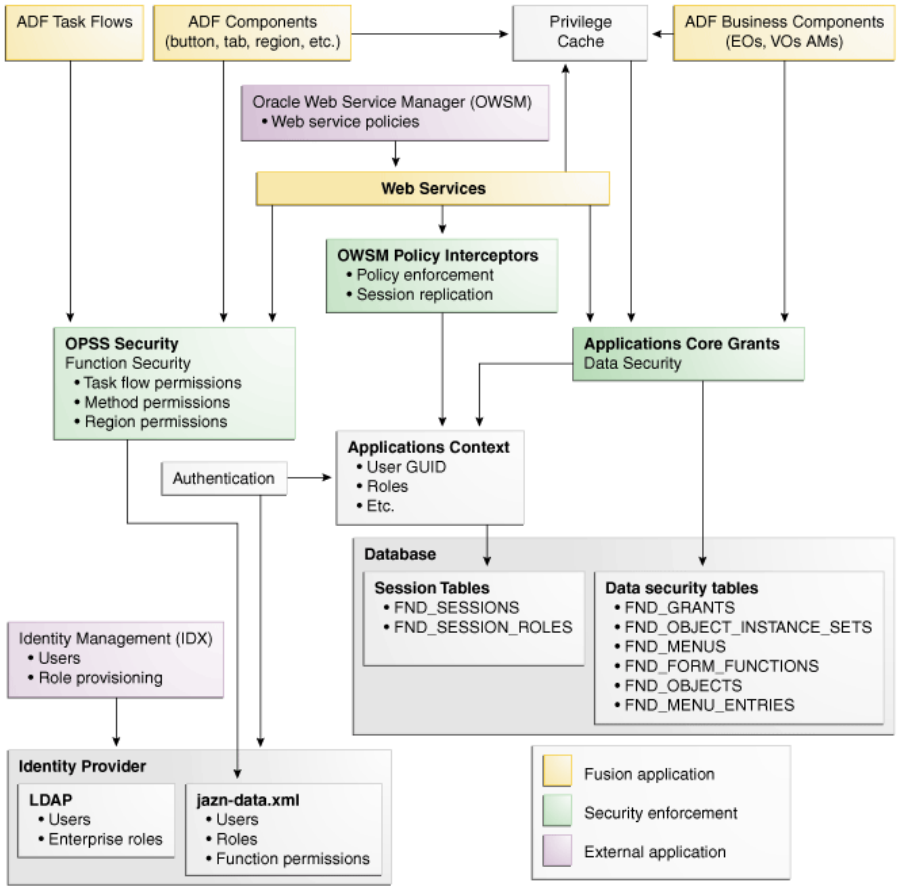
Additionally, for information about security and extending Oracle Fusion applications, see the *Oracle Fusion Applications Extensibility Guide*.

46.1.1 Architecture

Oracle Fusion applications are built on top of a fixed set of internal and third-party technologies. This set of technologies defines what may be used to develop, build, package, and run all Oracle Fusion applications. Each technology and component may have its own specific requirements for security implementation.

Figure 46–1 depicts the components in the Oracle Fusion Applications security approach.

Figure 46–1 Oracle Fusion Security Architecture



The Oracle Fusion Applications key technologies and features include:

- Application resources including ADF task flows, ADF components, and ADF Business Components. For more information, see:

  Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework.

- Oracle Platform Security Services (OPSS) security. For more information, see:

  Oracle Fusion Middleware Application Security Guide.

- Oracle Web Services Manager (Oracle WSM). For more information, see:

  Oracle Fusion Middleware Security and Administrator's Guide for Web Services.

- Oracle WSM policy interceptors. For more information, see Chapter 50, "Securing Web Services Use Cases."

- Identity providers including the LDAP-based provider and the file-based provider (`jazn-data.xml` file). For more information, see:

  Oracle Fusion Middleware Application Security Guide.

- Oracle Fusion application user sessions. For more information, see Chapter 47, "Implementing Application User Sessions."

- Oracle Fusion Data Security policies for data security. For more information, see Chapter 48, "Implementing Oracle Fusion Data Security."

- Oracle Fusion Applications security policies for function security. For more information, see Chapter 49, "Implementing Function Security."

Some of the technologies have an additional layer of security on top of the main security technologies.

46.1.1.1 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, 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. Figure 46–2 illustrates the layered architecture that combines these two security frameworks.

Figure 46–2 Oracle Platform Security Services Architecture

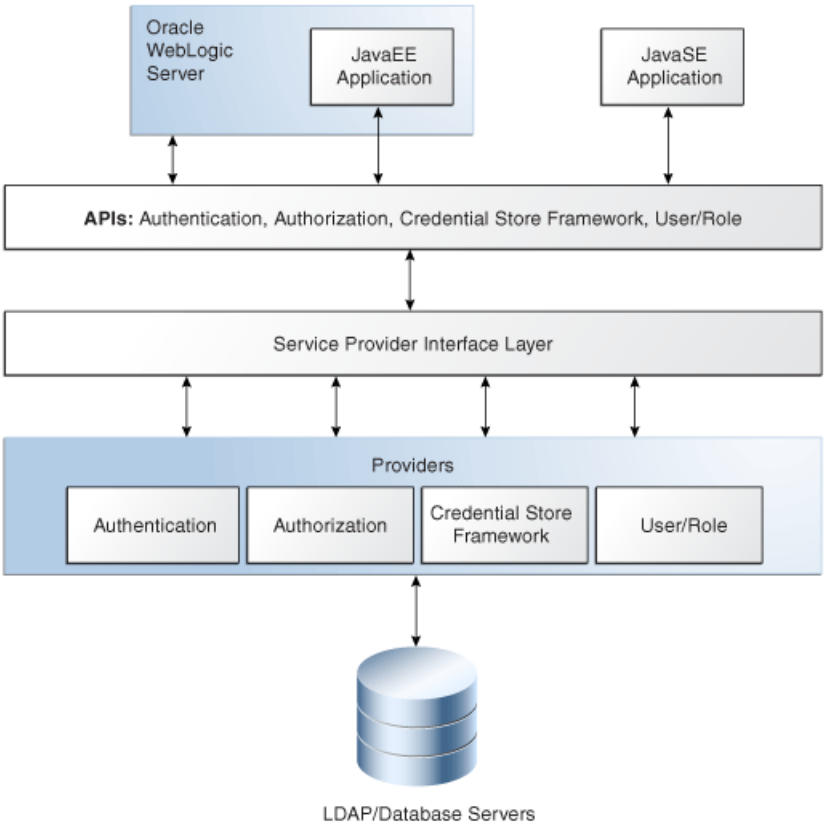


Figure 46–2 depicts the various security components as layers. The uppermost layer includes the Oracle WebLogic Server and the Java applications running on the server; under it, is the layer consisting of APIs for Authentication, Authorization, Credential Store Framework, User and Role, and identity 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 46–2, other providers include the role mapping and audit providers.

For more information about OPSS, see the "Understanding Security Concepts" part in the *Oracle Fusion Middleware Application Security Guide*.

The Security Service Provider Interface (SSPI) layer is accessed through OPSS APIs and provides Java EE container security in permission-based (JACC) mode and in resource-based (non-JACC) mode. It also provides resource-based authorization for the environment, thus allowing customers to choose their security model.

46.1.1.2 Oracle Web Services Manager

Oracle Web Services Manager (Oracle WSM) provides a policy framework to consistently secure Web services. Application developers attach policies using JDeveloper to the clients and services. Authentication and authorization are enforced on the services by Oracle WSM based on the policies attached to 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.

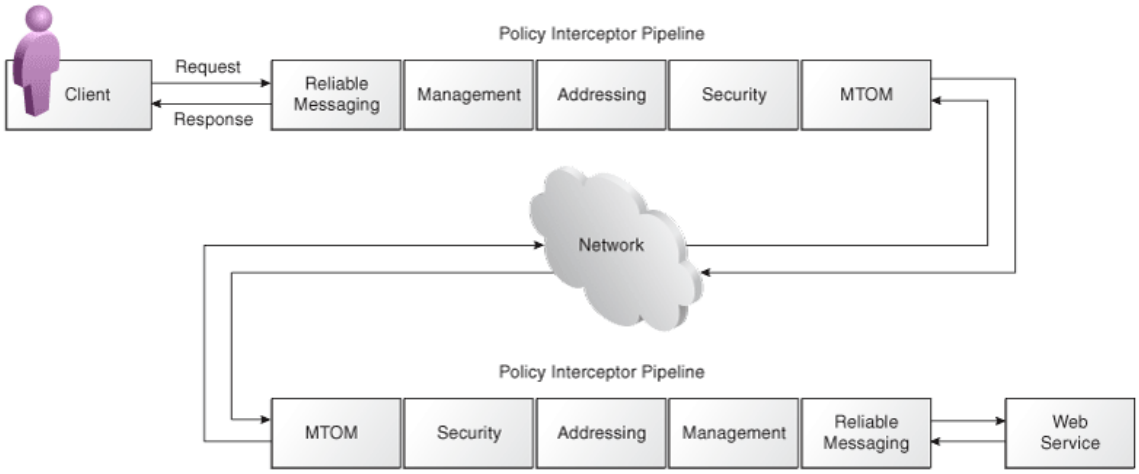
For more information about Oracle WSM, see *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

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 Figure 46–3.

Figure 46–3 Policy Interceptors Acting on Messages Between a Client and Web Service



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.

46.1.1.3 Oracle 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 provides declarative, permissions-based protection for ADF bounded task flows and top-level web pages that use ADF bindings.

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 definitions on the secured resources.

For more information about ADF Security, see the "Enabling ADF Security in a Fusion Web Application" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

46.1.1.4 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 Chapter 47, "Implementing Application User Sessions."

46.1.1.5 Oracle Fusion Data Security

Oracle Fusion Data Security implementation is a solution specifically for Oracle Fusion Applications. The security definitions for who can access what data are stored in the Oracle Fusion Data Security model in the Oracle Fusion Middleware schema. Developers create these definitions through SQL scripts, APIs or UIs. The definitions are then extracted and version controlled as seed data.

Oracle Fusion Data Security definitions are enforced either declaratively or programmatically in their application. In the context of an Oracle Fusion application, developers 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 Chapter 48, "Implementing Oracle Fusion Data Security."

46.1.1.6 Oracle Virtual Private Database

Oracle Virtual Private Database (VPD) enables controlling access to data on the database level using security policies associated with database objects. Use VPD when securing Personally Identifiable Information (PII) on the database level, for example.

For more information about VPD, see the "Using Oracle Virtual Private Database to Control Data Access" chapter in the *Oracle Database Security Guide*.

46.1.1.7 Oracle Data Integrator

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.

For more information about ODI, see the "Understanding Oracle Data Integrator" part in the *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator*.

46.1.2 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, Oracle Platform Security Services creates a subject containing the anonymous user `principal` and the anonymous role `role principal`. Oracle Platform Security Services 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.

46.1.2.1 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.

46.1.2.1.1 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.

46.1.2.1.2 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.

46.1.2.1.3 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.

46.1.2.1.4 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.

46.1.2.1.5 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 an IT security manager. By default, the deployment configuration disables data synchronization between the `jazn-data.xml` file and the LDAP server. This setting should not be modified.

46.1.2.1.6 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.

46.1.2.2 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

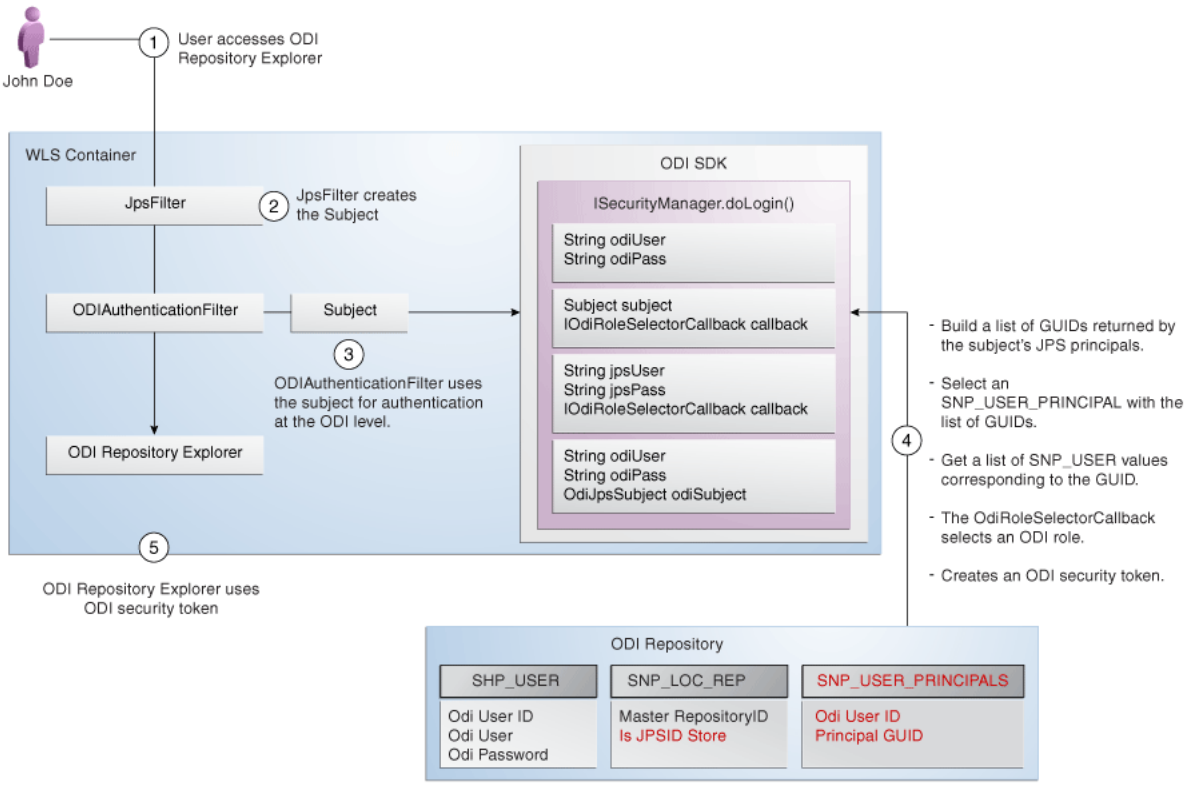
Oracle Fusion applications deploy Web Services for Remote Portlets (WSRP). As web services, they rely on Oracle Web Services Manager for Identity Propagation.

Oracle Data Integrator (ODI)

In a OPSS-enabled environment (Oracle Platform Security Services), ODI authentication happens in two phases.

Figure 46–4 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.

Figure 46–4 ODI Authentication



Audit Identity

When switching to the application identity, in order to preserve the submitting identity for auditing purposes, you must propagate the identity to be audited.

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. Example 46–1 illustrates the syntax of this method.

Example 46–1 Controlling which Identity Is Stored as the Audit Identity

```
ApplSession.setHistoryOverrideUserName()
```

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 Example 46–2.

Example 46–2 Storing the Original Identity

```
ApplSession session = ApplSessionUtil.getSession();
if (session != null)
session.setHistoryOverrideUserName(historyOverrideUserName);
```

46.1.2.3 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.

46.1.3 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.

46.1.3.1 OPSS Application Security Repository

The enterprise IT security manager 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 IT security managers 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.

For more information, see the "Enabling ADF Security in a Fusion Web Application" chapter in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

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 IT security manager.

46.1.3.2 Oracle Fusion Data Security Repository

Oracle Fusion Data Security repository is used to control access to data.

Policy Store Content

Enterprise IT security managers 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 Chapter 48, "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.

46.2 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.

46.2.1 APIs

You can implement authentication by using user and role APIs. For more information, see the "Developing with the User and Role API" chapter in the *Oracle Fusion Middleware Application Security Guide*.

46.2.2 Expression Language

You can use Expression Language to access security context information. 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.

For more information, see the "Enabling ADF Security in a Fusion Web Application" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

46.2.3 Non-browser Based Login

For information about using non-browser based security, see the "Securing Your Integrated Excel Workbook" chapter in the *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.

46.3 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.

46.3.1 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 an IT security manager 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 the application workspace.
3. Create an entitlement to group one or more ADF resources and their corresponding actions to entitle end users to access the resource.
4. Grant the entitlement to a custom duty role that was added to the Oracle Fusion application policy store.
5. Enable ADF Security for the application by running the Configure ADF Security wizard.

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 Chapter 49, "Implementing Function Security."

46.3.1.1 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 for 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 Chapter 49, "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.

For details about resource permissions and using APIs, see the "Understanding Security Concepts" part in the *Oracle Fusion Middleware Application Security Guide*.

46.3.1.2 Expression Language

You can use Expression Language to access security context information. Following are some useful expressions:

- `securityContext.taskflowViewable`
- `securityContext.regionViewable`
- `securityContext.userGrantedPermission`

For more information, see the "Enabling ADF Security in a Fusion Web Application" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

46.3.2 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 Chapter 48, "Implementing Oracle Fusion Data Security."

For more information about using Oracle Authorization Policy Manager, see the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

46.3.2.1 APIs and Expression Language

For information about using APIs and Expression Language to secure data, see Chapter 48, "Implementing Oracle Fusion Data Security."

46.3.2.2 Oracle Virtual Private Database

Oracle Virtual Private Database (VPD) enables creating security policies to control database access at the row and column level. Access is controlled at the database level. VPD adds a dynamic `WHERE` clause to a SQL statement issued against an 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. Using VPD affects performance. As such, make sure to evaluate your performance requirements prior to implementing the VPD solution.

For more information regarding VPD implementation, see the "Using Oracle Virtual Private Database to Control Data Access" chapter in the *Oracle Database Security Guide*.

46.3.2.3 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:

- Encryption and decryption APIs
- Oracle Fusion Data Security
- Row level Oracle Virtual Private Database

The security requirements for the PII attribute determine the technologies to be used.

46.3.2.4 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 "Data Security" chapter in the *Oracle Fusion Applications Security Guide*.

47

Implementing Application User Sessions

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:

47.1 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, and dictate what actions that user has.

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 Section 47.3, "Accessing Properties of the Applications Context."

47.2 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.

47.2.1 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.

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.

   Example 47–1 shows the `ApplSessionFilter` mapping definition to add.

Example 47–1 Creating a New Filter Mapping Definition

```
...
<filter-mapping>
   <filter-name>ApplSessionFilter</filter-name>
   <servlet-name>Faces Servlet</servlet-name>
   <dispatcher>FORWARD</dispatcher>
   <dispatcher>REQUEST</dispatcher>
</filter-mapping>
...
```

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 in order to change the ordering of the filter mapping, you must modify the *web.xml* file directly.

4. Save all changes.

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 Section 49.3, "Adding Function Security to the Application."

47.2.2 How to Configure the ADF Business Component Browser

The steps in Section 47.2.1, "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 Section 47.3.2.1, "Initializing Sessions."

47.2.3 What Happens at Runtime: How the Application User Session is Used

When you run your page, an authentication dialog displays. Login as *OPERATIONS* / *welcome1*. An application user session for the *OPERATIONS* user is created and associated with your application user session. Your page can access this application user session through the `ApplSessionUtil` class, as described in Section 47.3, "Accessing Properties of the Applications Context."

For details about enforcing security and granting access to application resources while testing the applications, see Section 49.3, "Adding Function Security to the Application."

47.3 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.applcore.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.
- Central context attributes.
- National Language Support (NLS) properties.

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 that are currently active. (Assigned to the currently logged in user).

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 a number of attributes they wish to store on the session. Developer 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.

47.3.1 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.fnd.applcore.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.

47.3.1.1 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. In order to use sessions, you must first configure your project to use application user sessions. For more information about configuring your project, see Section 47.2, "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 J2EE page.

For J2SE programs, such as JUnit tests, you must call an explicit API to initialize your session. As Example 47–2 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.

Example 47–2 Initializing Your Session

```
@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();
  }
```

Caution: Remember, every call to `initializeSession` should have a corresponding `terminateSession` invoked after the code completes.

47.3.1.2 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 Example 47–3 shows, after you complete those tasks you can access the session and its properties using the static APIs that are provided.

Example 47–3 Accessing the Session

```
ApplSession session = ApplSessionUtil.getSession();
String guid1 = session.getUserGuid();

String guid2 = ApplSessionUtil.getUserGuid();
```

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. Example 47–4 shows a mechanism for getting generic attributes.

Example 47–4 Getting Generic Attributes

```
String attr1 = ApplSessionUtil.getSessionAttribute("ATTRIBUTE1");
```

Example 47–5 shows the API you use to fetch attributes from a particular namespace.

Example 47–5 Fetching Attributes From A Particular Namespace

```
String attr1 = ApplSessionUtil,getNamespaceAttribute("MY$NAMESPACE",
"ATTRIBUTE1");
```

47.3.1.3 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 needs to be changed in the middle of the request.

Example 47–6 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.

Example 47–6 Setting Context Attributes

```
ApplSession session = ApplSessionUtil.getSession();
  if (session != null)
  {
    session.setProductFamily("FND");
    session.setSessionAttribute("ATTRIBUTE1", "VALUE1");
    session.setNamespaceAttribute("MY$NAMESPACE", "ATTRIBUTE1", "VALUE1");
  }
```

> **Note:** 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()`.

47.3.1.4 Accessing the Connection

The applications context does not hold onto connections, instead it obtains and releases them as needed. As Example 47–7 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 47–7 Accessing the Connection of the Current ApplSession

```
Connection conn = null;
ApplSession session = ApplSessionUtil.getSession();
if (session != null)
{
   try
   {
```

```
      conn = session.getConnection();
      ...
   }
   finally
   {
      if (conn != null)
      {
         session.releaseConnection(conn);
      }
   }
}
```

47.3.1.5 Accessing 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. As long as you are running from an environment where application user sessions are enabled, there should not be anything else you need to 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 with the exception of 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 Example 47–8, change it to return the value of the call to `ApplSessionUtil.getUserName()` in the `TestVORowImpl.java` that gets autogenerated.

Example 47–8 Changing the getUserName() Value

```
public String getUserName()
{
  return ApplSessionUtil.getUserName();
}
```

Whenever the `TestVO` view object is displayed, by default it will include the current user name field.

Example 47–9 uses the `SysadminInfo` field that was added to the `TestVO` view object to display a value when running the **FND** product.

Example 47–9 TestVO Example

```
public String getSysAdminInfo()
{
  String productName = ApplSessionUtil.getProduct();
  if ("FND".equals(productName))
  {
    return (String) getAttributeInternal(SYSADMININFO);
  }
```

```
  else
  {
    return null;
  }
}
```

47.3.2 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 Section 47.3, "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.

47.3.2.1 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 Example 47–10 shows, the lists must be of the same length.

Example 47–10 Initializing Sessions

```
DECLARE
  1_roleguids FND_TABLE_OF_VARCHAR2_4000 := FND_TABLE_OF_VARCHAR2_4000();
  1_rolenames FND_TABLE_OF_VARCHAR2_4000 := FND_TABLE_OF_VARCHAR2_4000();
BEGIN
  1_roleguids.extend(1);
  1_rolenames.extend(1);
  1_roleguids(1) := '1807EDD02DBB11DDBFDC91643D402C34';
  1_rolenames(1) := 'operationsRole';
  fnd_global.initialize_session('43B84790D5F011DCAF4F5FFD8462C8E7','OPERATIONS',
1_roleguids, 1_rolenames);
  <your code here>
  fnd_global.terminate_session;
END;
/
```

47.3.2.2 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.

47.3.2.3 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.

48

Implementing 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:

48.1 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.

> **Note:** Oracle Platform Security Services (OPSS) principal information is not stored in the Oracle Fusion Data Security schema. The OPSS principal may be stored in any third-party system. 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.

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 restrict the actions that are available for a given row.

Note: 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 Chapter 47, "Implementing Application User Sessions." 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 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.

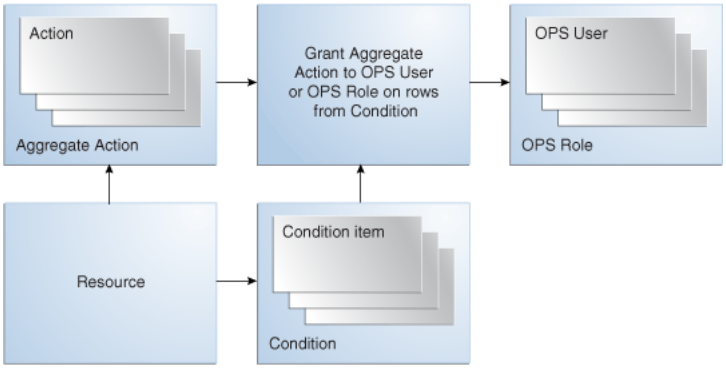
For additional information about Oracle Fusion Data Security, see the *Oracle Fusion Applications Security Guide*.

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 needs to 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. Figure 48–1 illustrates the logical data model implemented by Oracle Fusion Data Security.

Figure 48–1 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.

48.1.1 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 Section 48.4.2, "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) 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 are what is used to secure these events. An action must be associated with a resource.

> **Note:** The action name alone is not unique on the table; the combination of an action name and resource is what makes it unique.

Actions are stored in `FND_FORM_FUNCTIONS` table.

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.
- Once 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 Section 48.1.5, "Integrating Oracle Fusion Data Security with Virtual Private Database (VPD)".

Security Policy: 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`.

48.1.2 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. 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.

48.1.3 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 *Oracle Fusion Applications Common Implementation Guide*.

If data security task flows are used in a web application, that web application must be configured to use ADF Security in order 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 privilege cannot view the task flow. For more information about how to implement function security privileges and roles, see Chapter 49, "Implementing Function Security."

Table 48–1 lists the task flows and their parameters.

Table 48–1 Data Security Task Flows and Parameters

Task Flow Name	Task Flow XML	Parameters Passed	Behavior	Comments
Manage Database Resources	`/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/DBResourceTF.xml`	`module_id` (optional)	Goes to the Search page for database resources.	None.
Manage Database Resource	`/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/CreateDBResourceTF.xml`	`mode = edit` `dbResourceId`	Goes to the Edit page for a database resource.	None.

Manage Database Resource Conditions	`/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/CreateDBResourceTF.xml`	`mode = edit`
		`dbResourceId`
		`panelTab = conditions`
			Goes to the Conditions tab of the database resource Edit page.	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.

Manage Database Resource Actions	`/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/CreateDBResourceTF.xml`	`mode = edit`
		`dbResourceId`
		`panelTab = actions`
			Goes to the Actions tab of the database resource edit page.	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.

Manage Policy	`/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/PolicyTF.xml`	`grantGuid` (optional) `dbResourceId` (optional) `appId` (optional) `roleName` (optional)	This is the Create/Edit Policy page	There is no Search here, except to pick a specific database resource and pick a specific role.

48.1.4 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 Chapter 47, "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.

48.1.5 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 *policy* is enabled on a particular view or synonym that references that policy function. Policy functions based on fnd_data_security.get_security_predicate() are used to enforce data security rules.

To integrate with VPD:

1. Create an action.

   Create an action on the database resource that you want to secure. Using the Functions form, set the object column of the action to point to the data security object. This column is fnd_form_functions.object_id.

2. Create a view or synonym.

   Create a view or synonym with the exact same name as the action.

3. Add a policy.

   Add a policy in the database that will associate the policy function with the 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.

48.2 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 managing data security, see the "Managing Oracle Fusion Applications Data Security Policies" chapter in the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

48.2.1 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 48–2. The Oracle Fusion reference implementation predefines an *Application Developer* job role that inherits all the roles described in Table 48–2. It also seeds a user APPLICATION_DEVELOPER that inherits the *Application Developer* job role.

To use the standard Application Developer role:

- Login to Functional Setup Manager as the *APPLICATION_DEVELOPER* user. Contact the system administrator for the password.

If you have your own Product Family Administration role:

1. In Oracle Authorization Policy Manager (Oracle APM), 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 48–2.
3. Define the relationship between duty role and enterprise role.

Use cases:

- CRM Administrator logs in. He must be able to manage the CRM database resources. He must NOT be able to access the HCM resources.
- Super Administrator (Application Developer) logs in. He must be able to manage conditions for all Oracle Fusion resources.

Solution based on the above use cases:

1. Oracle Fusion Applications delivers the duty roles and policies as specified.
2. Oracle Authorization Policy Manager (APM) authors enterprise roles and maps the duty roles listed in Table 48–2. A security manager uses APM to define the relationship between a duty role and enterprise roles.
    - APM can create three enterprise roles, one per pillar. (Multiple product families can be included into one pillar).
    - APM can include the product family level duties into the above enterprise roles, as appropriate.
    - APM must ensure that they have the correct role GUIDs for the duty roles. (See the *jazn-data.xml* file.)
3. A security manager can create their own enterprise roles if required. They can determine which objects can be managed by specific data security administrators by including the duty roles into their custom enterprise roles. Security managers are expected to use the APM console to perform enterprise role to duty role mapping.
4. Policies runtime is based on role GUIDs only.

Reasoning:

Data security policies are made 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.

48.2.2 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 granted the appropriate data security manage privileges to your administrators.

Table 48–2 lists the duty roles that have been predefined by the Oracle Fusion security reference implementation to allow access to users to manage data security. These roles are administered at the product family level to manage resources and policies for that specific product family.

Table 48–2 Duty Roles to Manage Data Security Policies

#	Family	Duty Role
1	CRM	Customer Relationship Management Database Resource Administration Duty
2	HCM	Human Capital Management Database Resource Administration Duty
3	FSCM	Financials and Supply Chain Manufacturing Database Resource Management Duty
4	APM - CRM	APM - CRM Database Resource Administration Duty
5	APM - HCM	APM - HCM Database Resource Administration Duty
6	APM - FSCM	APM - FSCM Database Resource Administration Duty

48.3 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.

The authorization check is done automatically by ADF Business Components for standard operations, such as `read`, `update`, and `removeCurrentRow`. To perform a security check on non-standard operations you must call Oracle Fusion Data Security APIs directly.

You must identify your actions on the entity object. You cannot identify actions directly on an ADF Business Components view object; however, when a view object references an entity whose operations have been secured, the entity security policies also apply to the view object.

Oracle Fusion Data Security provides an implementation of a data security provider interface defined by ADF Business Components to perform the authorization check.

48.3.1 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 Example 48–1.

Example 48–1 Making the Oracle Fusion Data Security Provider the Data Security Provider

```
dataSecurityProviderClass=
  "oracle.apps.fnd.applcore.dataSecurity.util.FndDataSecurityProvider"
```

For example, the `adf-config.xml` file would contain the `sec:JaasSecurityContext` element definition shown in Example 48–2.

Example 48–2 sec:JaasSecurityContext Element Example

```
<sec:JaasSecurityContext intialContextFactoryClass=
  "oracle.adf.share.security.JAASIntialContextFactory"
jaasProviderClass="oracle.adf.share.security.providers.jps.JpsSecurity.Context"
dataSecurityProviderClass=
  "oracle.apps.fnd.applcore.dataSecurity.utl.FndDataSecurityProvider"
authorizationEnforce="true"
authenticationRequire="true"/>
```

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 once 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. 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 Section 48.3.4, "How to Perform Authorization Checks for Custom Operations."

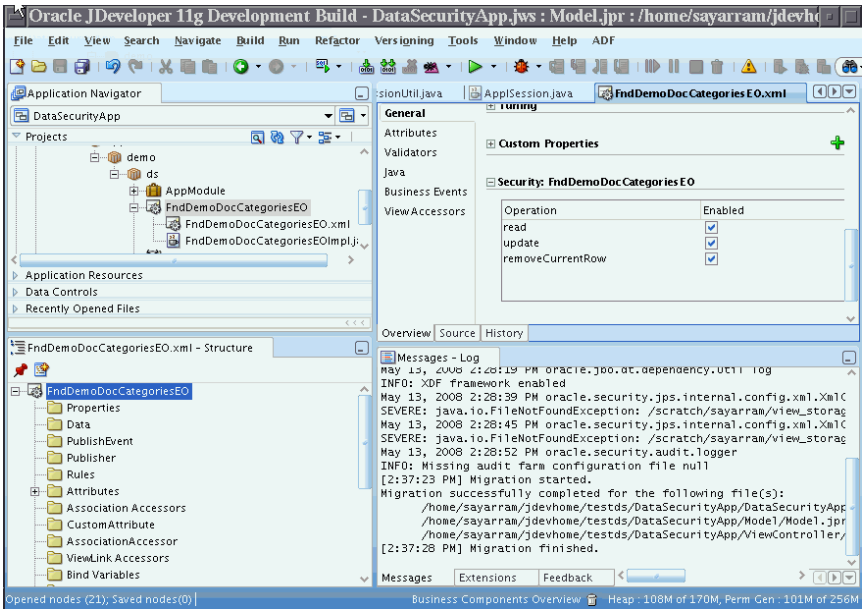
There may be other reasons to invoke Oracle Fusion Data Security APIs manually to determine the SQL predicate for a given action or to do an authorization check. For example, you might query VPD policies written based on `fnd_data_security.getSecurityPredicate()` to enforce data security rules.

48.3.2 How to Secure Rows Queried By Entity-Based View Objects

At design time, you can identify the various operations on a given entity object to be secured by using the entity's overview editor and going to the Security section, as shown in Figure 48–2. The overview editor exposes a set of standard operations (`read`, `update`, `removeCurrentRow`) as checkboxes that you can select. Based on the operations that you select, the appropriate checks are done at runtime.

This means that when you define actions in Oracle Fusion Data Security, the actions for those objects should be named as `read`, `update`, or `delete` to correspond to the entity object security operations that get enabled.

Figure 48–2 Entity Object Overview Editor — Security Section



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 needs to be done, you must use a custom action.

Note: The default Oracle Fusion Data Security provider implementation assumes that the object name in `FND_OBJECTS` for the entity being secured is the database table/view name backing this entity. If the entity is a translatable entity (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 is not sufficient, one can set a custom property on the entity object to identify an object name from the Oracle Fusion Data Security repository that should be used to secure this entity. The custom property `OA_DS_BASE_TABLE` should be set to accomplish this.

At runtime, for the read operation, ADF Business Components automatically invokes the Oracle Fusion Data Security Provider (which is registered with ADF Business Components in `adf-config.xml` when you secure the read operation on the entity object), to identify the `WHERE` clause (if any) that needs to be added to the SQL statement for the entity object. This is done prior to executing the query.

Once the query has been executed, ADF Business Components invokes the Oracle Fusion Data Security Provider again to perform the authorization check for standard operations, (`update` and `removeCurrentRow`), to see if the user has update and delete access to that row.

In the case of custom privilege that you define, you must create a view criteria and apply it to the view instance that you want the application module data model to filter. In either case, the user must have sufficient privileges to view the filtered rows. The

action name that you define in Oracle Fusion Data Security must match the custom action specified on the entity object.

To secure the rows displayed to a user for read privilege:

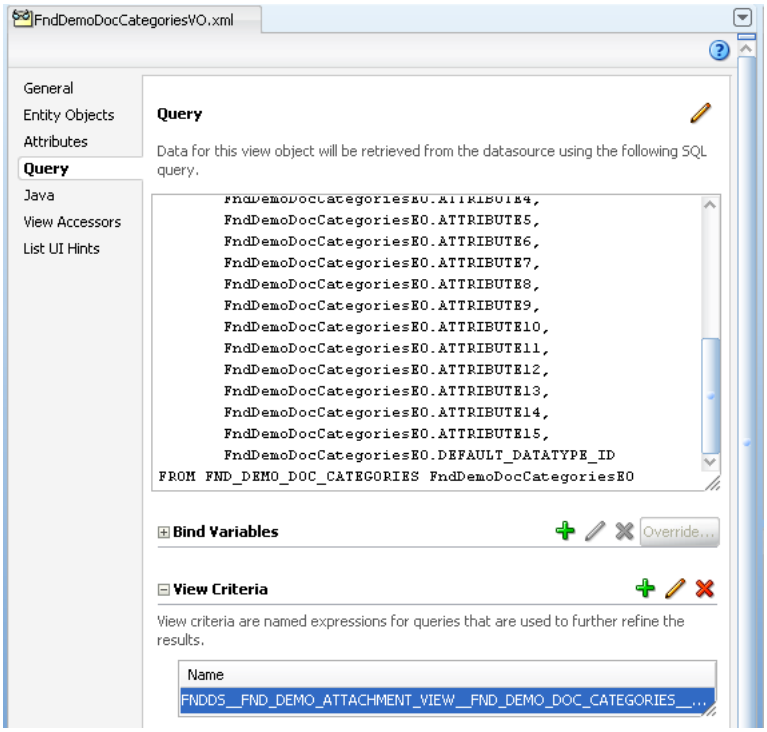
1. In the Application Navigator, double-click the entity object.
2. In the overview editor for the entity object, click the **General** navigation tab and expand the **Security** section.
3. In the Security section, select the **read** action.

To secure rows displayed to a user based on a custom privilege:

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 **Add New View Criteria** to add a dummy view criteria.

   Figure 48–3 shows a dummy view criteria that has been added in the overview editor for the view object.

Figure 48–3 View Object Overview Editor — View Criteria Selection



3. In the Edit View Criteria dialog, create a dummy view criteria with no view criteria items and name the view criteria using the following format:

   `FNDDS__privilegeName__objectName__objectAlias`

   where:

   `privilegeName` is the privilege name that is used to filter the data.

   `objectName` is the name of the secured database resource in the `FND_OBJECTS` table.

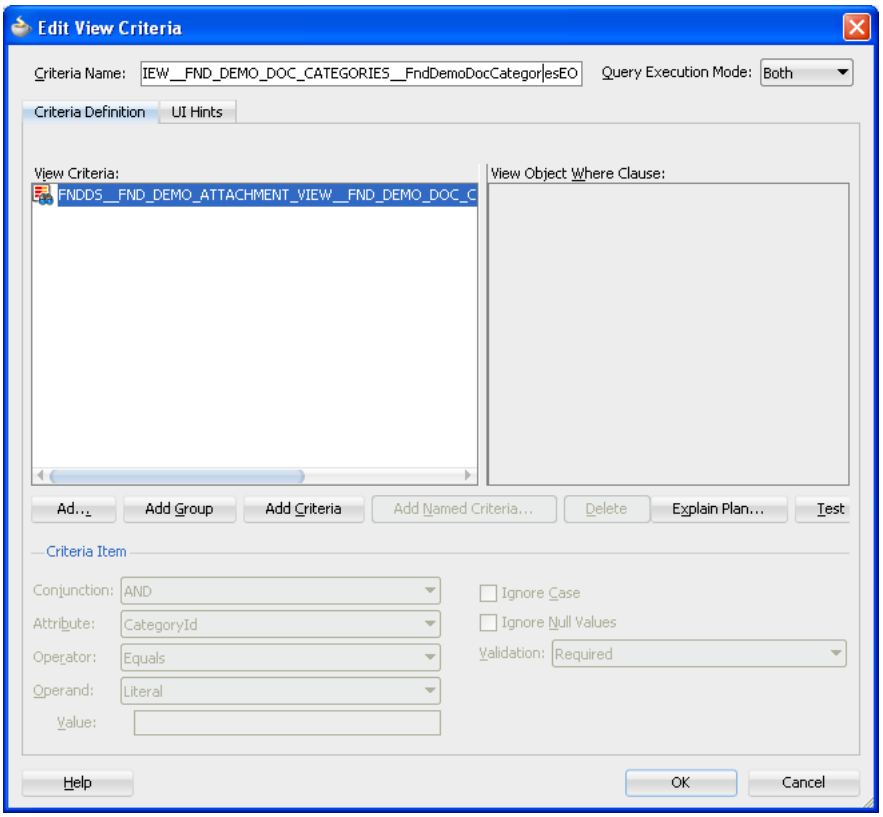
`objectAlias` is optional and is the alias for the resource.

> **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 Figure 48–4.

   The query execution mode for the dummy 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.

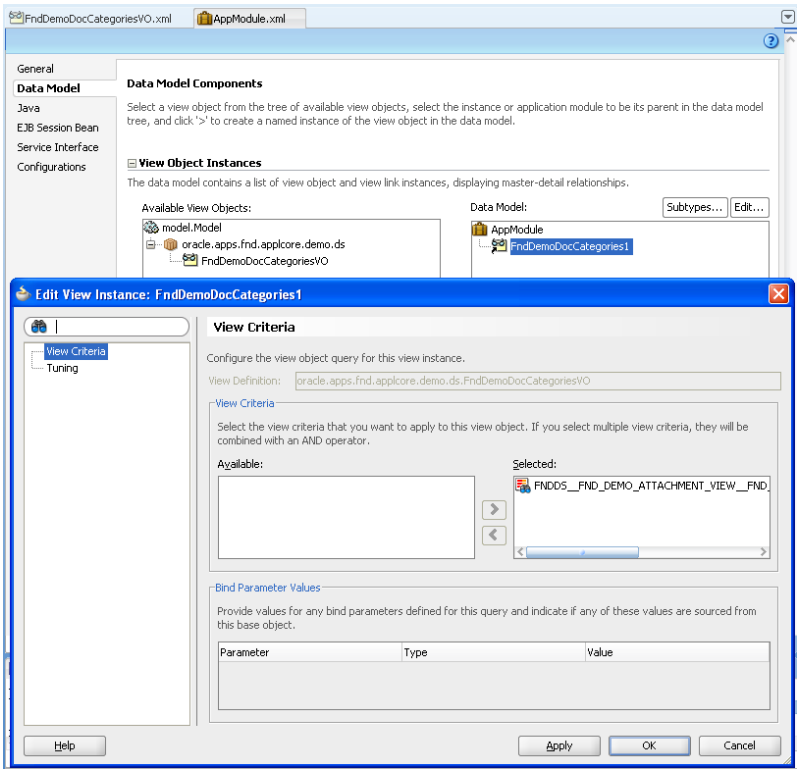
Figure 48–4 Edit View Criteria Dialog



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 Figure 48–5.

   Alternatively, you can apply the view criteria at runtime in your code by calling the view object's `applyViewCriteria(viewCriteriaName)` API.

Figure 48–5 Application Module Overview Editor — Edit View Instance Dialog



48.3.3 What Happens at Runtime: How Oracle Fusion Data Security Filters View Instance Rows

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 in order to 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 privilege is applied, the instance sets corresponding to that privilege are OR'ed. When multiple privilege checks are applied, the instance sets of a privilege are AND'ed with the instance sets of another privilege. For example, for an object, for privilege `priv1`, the user has instance sets IS1, IS2. For the same object for privilege `priv2`, the user has instance sets IS3, IS4. If both `priv1` and `priv2` checks are applied simultaneously (using view criteria), the `WHERE` clause would be `(IS1 OR IS2) AND (IS3 OR IS4)`.

48.3.4 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 Example 48–3.

Example 48–3 Manually Invoking the Authorization Checking API

```
if(row.getSecurityHints().allowsOperation("ApprovePO").hasPermission())
        // code for PO approval
else
        // display error message
```

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.

48.3.5 How to Test Privileges Using Expression Language Expressions in the User Interface

You can test data security actions for standard (read, update, delete) or custom actions 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 action. The transient attribute can be used to control the rendering of some other attribute in the read-only view object on the page.

- Example 48–4 shows the Expression Language expression to use on ADF bindings for view object attributes.

Example 48–4 Using Expression Language on View Object Attributes

```
#{bindings.<attrName>.hints.allows.<privilegeName>}
```

For example, your UI might have a button to control the grant for the `UpdateEmployeeSalary` privilege; the Expression Language expression on the button may be defined as shown in Example 48–5.

Example 48–5 Correct Expression Language Expression Example

```
#{bindings.PersonId.hints.allows.UpdateEmployeeSalary}
```

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` privilege. 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 Example 48–6 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 Example 48–5 with a custom action.

Example 48–6 Incorrect Expression Language Expression Example

```
#{bindings.<attrName>.hints.<attrName>.allows.<privilegeName>}
```

- Example 48–7 shows the expression to use for table iterators:

Example 48–7 Using Expression Language for Table Iterators Example

```
#{row.hints.allows.<privilegeName>}
```

Do not use the expression shown in Example 48–8 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 Example 48–7 with a custom action.

Example 48–8 Incorrect Expression Language Expression Example

```
#{row.hints.<attrName>.allows.<privilegeName>}
```

- 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 Example 48–9 if the view object is an updateable view object based on an entity object.

Example 48–9 Groovy Expression Example

```
object.getSecurityHints().allowsOperation("updateCategory").hasPermission()
```

- When using Groovy expressions, use the expression shown in Example 48–10 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.

Example 48–10 Groovy Expression if View Object is Read-Only Example

```
oracle.apps.fnd.applcore.dataSecurity.dataSecurityService.applicationModule.DataSe
curityAMImpl.
testPrivilege("VIEW_PERSON_NAME_LIKE_T","PER_EL_TEST_PERSONS",
PersonId.toString(),
null,null,null,null,object.getViewObject().getDBTransaction());
```

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.

Example 48–11 shows an EL expression to conditionally render the UI component for the `Gender` attribute:

Example 48–11 Implementing Attribute Security Example

```
<af:panelFormLayout id="pfl1" partialTriggers="it6 cb3 cb1 cb4 cb5">
<af:inputText value="#{bindings.Gender.inputValue}"
              label="#{bindings.Gender.hints.label}"
              required="#{bindings.Gender.hints.mandatory}"
              columns="#{bindings.Gender.hints.displayWidth}"
              maximumLength="#{bindings.Gender.hints.precision}"
              shortDesc="#{bindings.Gender.hints.tooltip}" id="it4"
              rendered="#{bindings.TransientGenderAttr}">
  <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>
<f:facet name="footer">
<af:panelGroupLayout layout="vertical" id="pgl1">
  <af:panelGroupLayout layout="horizontal" id="pgl2">
    <af:commandButton actionListener="#{bindings.First.execute}"
                      text="#{applcoreBundle.FIRST}"
                      disabled="#{!bindings.First.enabled}"
                      partialSubmit="true" id="cb3"/>
    <af:commandButton actionListener="#{bindings.Previous.execute}"
                      text="#{applcoreBundle.PREVIOUS}"
                      disabled="#{!bindings.Previous.enabled}"
                      partialSubmit="true" id="cb1"/>
    <af:commandButton actionListener="#{bindings.Next.execute}"
                      text="#{applcoreBundle.NEXT}"
                      disabled="#{!bindings.Next.enabled}"
                      partialSubmit="true" id="cb4"/>
    <af:commandButton actionListener="#{bindings.Last.execute}"
                      text="#{applcoreBundle.LAST}"
                      disabled="#{!bindings.Last.enabled}"
                      partialSubmit="true" id="cb5"/>
  </af:panelGroupLayout>
    <af:commandButton text="#{applcoreBundle.SUBMIT}" id="cb2"/>
  </af:panelGroupLayout>
</f:facet>
</af:panelFormLayout>
```

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.

48.4 Using Oracle Fusion Data Security to Secure New Business Resources

The general process for defining Oracle Fusion Data Security policies to secure business resources that you add to your Oracle Fusion application is as follows.

1. Identify the business resource 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 resource:

   Tip: Conditions may be static or parameterized. For details about parameterizing conditions, see Section 48.4.2, "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 resource:
   - 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 at this time.

      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.

48.4.1 How to Use Oracle Fusion Data Security to Secure a Business Object

This example shows how to secure a document categories business resource. The document categories business data is stored in the `FND_DEMO_DOC_CATEGORIES` table. Table 48–3 lists the document category definitions for the table.

Table 48–3 FND_DEMO_DOC_CATEGORIES Table Definition

Name	Value
CATEGORY_ID	NOT NULL NUMBER
APPLICATION_ID	NUMBER
CREATION_DATE	NOT NULL DATE
CREATED_BY	NOT NULL NUMBER
LAST_UPDATE_DATE	NOT NULL DATE
LAST_UPDATED_BY	NOT NULL NUMBER
LAST_UPDATE_LOGIN	NUMBER
NAME	NOT NULL VARCHAR2(30)
START_DATE_ACTIVE	DATE
END_DATE_ACTIVE	DATE
ATTRIBUTE_CATEGORY	VARCHAR2(30)
ATTRIBUTE1 thru ATTRIBUTE15	VARCHAR2(150)
DEFAULT_DATATYPE_ID	NUMBER

Before you begin:

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 Section 47.2, "Configuring Your Project to Use Application User Sessions."

You must enable security on your application. For more information about enabling security, see Section 49.3.5, "How to Enforce Authorization for Securable ADF Artifacts."

To secure a business object:

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.
   - 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.

48.4.2 How to Use Parameterized Conditions When Securing a Business Object

Parameterized conditions allow conditions to be specified generally but granted specifically. Parameterized conditions should be used whenever possible 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.

Example 48–12 shows how a parameterized condition can be reused.

Example 48–12 Reusing Parameterized Conditions

```
OIS1. Employees in a particular region.
    Predicate: "&TABLE_ALIAS.REGION = &GRANT_ALIAS.PARAMETER1"
```

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 in order to keep possible duplicate column names from causing collisions.

48.4.2.1 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 `'FM99999999999999999999.99999999999999999999'`.)

  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.

48.4.2.2 Writing Performance Type Conversions in Predicates

Use the following rules when writing performance type conversions in your predicates.

Integer Equality:

Example 48–13 shows the correct Integer equality format to use.

Example 48–13 Integer Equality Format

```
to_char(&TABLE_ALIAS.ITEM_ID) = &GRANT_ALIAS.PARAMETER1
```

Example 48–14 shows Integer equality formats that you should NOT use.

Example 48–14 Incorrect Integer Equality Formats

```
&TABLE_ALIAS.ITEM_ID = to_number(&GRANT_ALIAS.PARAMETER1)
&TABLE_ALIAS.ITEM_ID = fnd_data_security.to_int(&GRANT_ALIAS.PARAMETER1)
```

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:

Example 48–15 shows the correct Integer range format to use.

Example 48–15 Integer Range Format

```
&TABLE_ALIAS.LEVEL >= fnd_data_security.to_init(&GRANT_ALIAS.PARAMETER1)
```

Example 48–16 shows Integer range formats that you should NOT use.

Example 48–16 Incorrect Integer Range Formats

```
&TABLE_ALIAS.LEVEL >= to_number(&GRANT_ALIAS.PARAMETER1)
to_char(&TABLE_ALIAS.LEVEL) >= &GRANT_ALIAS.PARAMETER1
```

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:

Example 48–17 shows the correct Float equality format to use.

Example 48–17 Float Equality Format

```
to_char(&TABLE_ALIAS.TEMP,'FM99999999999999999999.9999999999999999999') =
&GRANT_ALIAS.PARAMETER1
```

Example 48–18 shows Float equality formats that you should NOT use.

Example 48–18 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)
```

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:

Example 48–19 shows the correct Float range format to use.

Example 48–19 Float Range Format

```
&TABLE_ALIAS.TEMP > fnd_data_security.to_decimal(&GRANT_ALIAS.PARAMETER1)
```

Example 48–20 shows Float range formats that you should NOT use.

Example 48–20 Incorrect Float Range Formats

```
to_char(&TABLE_ALIAS.TEMP,'FM99999999999999999999.99999999999999999999') >
&GRANT_ALIAS.PARAMETER1
to_char(&TABLE_ALIAS.TEMP) > &GRANT_ALIAS.PARAMETER1
&TABLE_ALIAS.TEMP > to_number(&GRANT_ALIAS.PARAMETER1)
&TABLE_ALIAS.TEMP > fnd_data_security.to_init(&GRANT_ALIAS.PARAMETER1)
```

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:

Example 48–21 shows the correct Date equality format to use.

Example 48–21 Date Equality Format

```
to_char(&TABLE_ALIAS.ACTION_DATE,'YYYY/MM/DD HH24:MI:SS') = (&GRANT_
ALIAS.PARAMETER1)
```

Example 48–22 shows Date equality formats that you should NOT use.

Example 48–22 Incorrect Date Equality Formats

```
to_char(&TABLE_ALIAS.ACTION_DATE) = (&GRANT_ALIAS.PARAMETER1)
&TABLE_ALIAS.ACTION_DATE = fnd_data_security.to_date(&GRANT_ALIAS.PARAMETER1)
&TABLE_ALIAS.ACTION_DATE = to_date(&GRANT_ALIAS.PARAMETER1)
```

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:

Example 48–23 shows the correct Date range format to use.

Example 48–23 Date Range Format

```
to_char(&TABLE_ALIAS.HIRE_DATE,'YYYY/MM/DD HH24:MI:SS') > (&GRANT_
ALIAS.PARAMETER1)
```

Example 48–24 shows Date range formats that you should NOT use.

Example 48–24 Incorrect Date Range Formats

```
&TABLE_ALIAS.HIRE_DATE > fnd_data_security.to_date(&GRANT_ALIAS.PARAMETER1)
to_char(&TABLE_ALIAS.HIRE_DATE) > (&GRANT_ALIAS.PARAMETER1)
&TABLE_ALIAS.HIRE_DATE > to_date(&GRANT_ALIAS.PARAMETER1)
```

In this case, you can use `to_char()` with a format mask because the canonical format maintains proper ordering of the character domain.

48.4.3 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.

Example 48–25 shows an example of the identity store in the flat file used by JDeveloper.

Example 48–25 Oracle Platform Security Services (OPSS) XML Policy Store

```
<?xml version="1.0" encoding="windows-1252" standalone="yes" ?>
<jazn-data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://xmlns.oracle.com/oraclass/schema/jazn-data-1
1_0.xsd">
<jazn-realm default="jazn.com">
  <realm>
    <name>jazn.com</name>
    <users>
      <user>
        <name>admin</name>
        <credentials>{903}Qh3td3z2HHlun9ROrbvE6bYJDnNaoGir</credentials>
      </user>
      <user>
        <name>joeUser</name>
        <credentials>{903}FarN3p4C9IU9PZODN5RLmrpHf45eCw+W</credentials>
      </user>
    </users>
    <roles>
      <role>
        <name>adminRole</name>
        <members>
          <member>
            <type>user</type>
            <name>admin</name>
          </member>
        </members>
      </role>
      <role>
        <name>regularUserRole</name>
        <members>
          <member>
            <type>user</type>
            <name>admin</name>
          </member>
          <member>
            <type>user</type>
            <name>joeUser</name>
          </member>
```

```
        </members>
      </role>
    </roles>
  </realm>
</jazn-realm>
</jazn-data>
```

48.4.4 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 "Securing Oracle Fusion Applications" chapter in the *Oracle Fusion Applications Administrator's Guide*.

48.5 Getting Security Information from the Application User Session Context

Oracle Fusion Data Security is part of Oracle Platform Security Services. Hence data security information can be obtained from security context as required while developing the application.

48.5.1 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 Example 48–26.

Example 48–26 Check Action Method

```
/**
 * 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.
 * If null, then no data context is used.
 * @return true if privilege is accessible, false otherwise.
 */
public boolean testPrivilege(Privilege privilege, DataContext context)
```

Where `DataContext` is a container class that represents the resource and optionally the primary keys of that resource for which data security check needs to 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 Example 48–27.

Example 48–27 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 predicat.
 * @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)
```

- Get all the actions available on a given row instance for the current user, as shown in Example 48–28.

Example 48–28 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)
```

- Get all the aggregate actions available on a given row instance for the current user, as shown in Example 48–29.

Example 48–29 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)
```

As shown in Example 48–30, static APIs are also provided in the `DataSecurityAMImpl` class for the above core methods that take in a `DBTransaction` object.

Example 48–30 Static API Examples

```
public static boolean testPrivilege(Privilege privilege, DataContext context,
      DBTransaction dbTxn)

public static String getSecurityPredicate(Privilege privilege,
     DataContext dataContext, String grantInstanceType, String statementType,
```

```
    String tableAlias, DBTransaction dbTxn)

public static String[] getAggregatePrivileges(DataContext dataContext,
     DBTransaction dbTxn)
public static String[] getPrivileges(DataContext dataContext, DBTransaction dbTxn)
```

48.5.2 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 Example 48–31. The user is determined from the user session context.

Example 48–31 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 NUMBER 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 NUMBER DEFAULT NULL,
  p_instance_pk3_value   IN NUMBER DEFAULT NULL,
  p_instance_pk4_value   IN NUMBER DEFAULT NULL,
  p_instance_pk5_value   IN NUMBER DEFAULT NULL
)RETURN VARCHAR2;
```

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 Example 48–32. The user is determined from the user session context.

Example 48–32 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,
```

```
  /* Name of action. (Optional) */
  /* 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',
  /* (Optionsl) statement type: 'OTHER', 'VPD', 'EXISTS' = to check existence. */
  x_predicate            OUT NOCOPY VATRCHAR2,
  x_return_status        OUT NOCOPY VARCHAR2,
  p_table_alias          IN VARCHAR2 DEFAULT NULL
  /* (Optional) */
);
```

`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`.

48.6 Understanding Data Security Performance Best Practices

The `WHERE` clause associated with a given resource is constructed by doing an `OR` of all the predicates associated with the conditions granted to a user or role. Therefore, it is important that the predicate for a condition be efficient so that it returns as small a number of rows as possible and it also makes use of an index.

Data security should only be used when the combined predicates (Applications Code + Data Security Predicates) are efficient, and return only relevant rows. Most queries should only return either one row or just a few rows. The maximum number of rows that data security should consider operating on is 100 rows. If you are seeing performance problems with queries that apply data security to more than 100 rows, the query needs to be made more selective. Blind queries against tables secured with data security are not allowed.

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?"*

48.7 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.

Note: 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 "Standard Diagnostic Testing Administration Tasks and Tools" section in the *Oracle Fusion Applications Administrator's Guide*.

48.7.1 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 scripts 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 LDA.
- 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.

```
>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:

```
>wls:/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/system11.1.1.4.37.56.69/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 Section 48.7.2, "How to Validate Applications Context."

48.7.2 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.

48.8 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.

48.8.1 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.

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.
3. 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.
4. Create a managed bean and define an initialization method to populate the input parameters for the data security task flows, as shown in Table 48–4.

   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.
5. 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.
6. Enable function security grants to be made by the data security task flows.
7. Grant view permission to an application role that allows the end user to access the task flow.
8. Configure the application to access the domain LDAP policy store.

Table 48–4 describes the input parameters that your managed bean must initialize for each task flow.

Table 48–4 Data Security Task Flows and Their Input Parameters

Task Flow Name	Task Flow XML	Parameters Passed	Behavior

Object-centric task flow (also referred to as (`ObjectLevelTF`)	`/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/ObjectLevelFT.xml`	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`
			Create and manage grants to multiple application roles for a single business object.

Role-centric task flow (also referred to as `ProfileTF`)	`/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/ProfileTF.xml`	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`
			Create and manage grants to a single application role profile for multiple business objects.

Table 48–4 (Cont.) Data Security Task Flows and Their Input Parameters

Task Flow Name	Task Flow XML	Parameters Passed	Behavior

Instance-level task flow (also referred to as `ObjectInstTF`)	`/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/ObjectInstTF.xml`	Specify the name of the parent object from which the object instance will be fetched:
		`objectName`
		Specify the primary key of the object instance for which grants will be shared:
		`instancePk1`
		`instancePk2`
		`instancePk3`
		`instancePk4`
		`instancePk5`
		Specify the list of securable actions to be displayed in the UI for the object:
		`actions`
		Specify the ID of the customized task flow:
		`taskflowId`
		Specify the holder of the customized task flow parameters:
		`parameterMap`
			Confer existing grants for a single instance of a business object to another user (or user group).

Role management task flow (also referred to as `RoleManagementTF`)	`/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/RoleManagementTF.xml`	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`
			Create and edit custom application roles.

48.8.2 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.

48.8.2.1 Creating 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.

Example 48–33 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.

Example 48–33 ObjectLevelTF Reference in Calling Task Flow Configuration File

```
<task-flow-call id="ObjectLevelTF">
 <task-flow-reference>
   <document>/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/
                                                    ObjectLevelTF.xml</document>
   <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 48–34 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.

Example 48–34 ProfileTF Reference in Calling Task Flow Configuration File

```
<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 48–35 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.

Example 48–35 RoleManagementTF Reference in Calling Task Flow Configuration File

```
<task-flow-call id="RoleManagementTF">
  <task-flow-reference>
    <document>/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/
                                   taskflow/RoleManagementTF.xml</document>
    <id>RoleManagementTF</id>
  </task-flow-reference>
  <input-parameter>
    <name>roleCategories</name>
    <value>#{pageFlowScope.roleCategories}</value>
  </input-parameter>
  <input-parameter>
    <name>title</name>
    <value>#{pageFlowScope.title}</value>
  </input-parameter>
</task-flow-call>
```

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 Section 48.8.2.2, "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 48–33.
   - If you dropped the role-centric task flow (the activity references task flow `<id>ProfileTF</id>`), add the input parameter values from Example 48–34.
   - If you dropped the role management task flow (the activity references task flow `<id>RoleManagementTF</id>`), add the input parameter values from Example 48–35.

   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.

48.8.2.2 Initializing 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.

Example 48–36 shows the additional source code that your managed bean must include to initialize the object-centric (ObjectLevelTF) task flow.

Example 48–36 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 action1
 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 paramters in pageflow scope
 AdfFacesContext context = AdfFacesContext.getCurrentInstance();
 Map pageFlowScope = context.getPageFlowScope();
 pageFlowScope.put("objectName", objectName);
 pageFlowScope.put("roleCategories",roleCategories);
 pageFlowScope.put("actions", actions);
 pageFlowScope.put("disableViewAll", false);
 pageFlowScope.put("disableUpdateAll", false);
```

Example 48–37 shows the additional source code that your managed bean must include to initialize the role-centric (ProfileTF) task flow.

Example 48–37 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 action1
 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 48–38 shows the additional source code that your managed bean must include to initialize the role management (RoleManagementTF) task flow.

Example 48–38 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");
```

Before you begin:

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.

Example 48–39 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 Section 48.8.2.3, "Registering the Managed Bean with Your Application's Task Flow."

Example 48–39 Button Component with Action Attribute to Invoke Bean Method

```
<af:commandButton text="Manage Roles" id="cb1"
            action="#{backingBeanScope.roleManageBean.initializeRoleManagement}"/>
```

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 48–36.
   - If you want to initialize the role-centric task flow (also called the Profile task flow), add the input parameter property declarations from Example 48–37.
   - If you want to initialize the role management task flow, add the input parameter property declarations from Example 48–38.

   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 Section 48.8.2.1, "Creating a Task Flow Call Activity in Your Application's Task Flow."

48.8.2.3 Registering 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. Example 48–40 uses the bean identifier `roleManageBean` to match the **Action** attribute definition specified on the button component shown in Example 48–39.

Example 48–40 Managed Bean Declaration in Calling Task Flow Configuration File

```
<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>
```

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 Example 48–40.

48.8.3 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.

48.8.3.1 Creating 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 Example 48–42.

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.

Example 48–41 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.

Example 48–41 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>
```

Before you begin:

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. When the end user clicks the button, the button will invoke the show popup operation. 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.

Example 48–42 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.

Example 48–42 Button Component with Show Popup Operation to Invoke Dialog

```
...
<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>
```

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 Section 48.8.2.2, "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 48–33.
   - If you dropped the role-centric task flow (activity will show task flow reference `<id>ProfileTK</id>`), add the input parameter values from Example 48–34.
   - If you dropped the role management task flow (activity will show task flow reference `<id>RoleManagementTF</id>`), add the input parameter values from Example 48–35.

   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.

48.8.3.2 Initializing 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.

Example 48–43 shows additional source code that your managed bean must include to initialize the object-instance (`ObjectInstlTF`) task flow.

Example 48–43 Source for Populating the ObjectInstTF Input Parameters

```
//Source for manaaged 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");
   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 48–44 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 Example 48–42.

Example 48–44 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
}
```

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 Example 48–42).

3. In the method definition, create the input parameter property definitions by copying and pasting from Example 48–43 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 Example 48–42).

6. In the method definition, create the save operation by copying and pasting from Example 48–44 into the new save grants method definition.

   Copying from the sample ensures the method `saveInstanceGrants()` is called exactly as shown.

48.8.3.3 Registering 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. Example 48–45 uses the bean identifier `objectInstanceBean` to match the bean references specified in the listener properties shown in Example 48–42.

Example 48–45 Managed Bean Definition in Calling Task Flow Configuration File

```
<adfc-config xmlns="http://xmlns.oracle.com/adf/controller" version="1.2">
   <task-flow-definition id="MyCallingTaskFlow">
     <default-activity id="__1">CallObjectInstTF</default-activity>
     <managed-bean id="__4">
        <managed-bean-name id="__5">objectInstanceBean</managed-bean-name>
        <managed-bean-class id="__2">
                  oracle.apps.fnd.applcore.dataSecurity.view.DSObjectInstance</managed-bean-class>
        <managed-bean-scope id="__3">backingBean</managed-bean-scope>
     </managed-bean>
   </task-flow-definition>
   ...
</adfc-config>
```

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 Example 48–45.

48.8.4 How to Grant the End User Access to the Data Security Task Flows

The data security task flows, once 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. Example 48–46 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/applcore/dataSecurity/ui/taskflow/.`

Example 48–46 Grant to View Data Security Task Flows

```
<grant>
   <grantee>
      ...
   </grantee>
   <permissions>
      <permission>
        <class>oracle.adf.controller.security.TaskFlowPermission</class>
        <name>/WEB-INF/oracle/apps/fnd/applcore/dataSecurity/ui/taskflow/.*</name>
        <actions>view</actions>
      </permission>
   </permissions>
</grant>
```

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 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 Example 48–46 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.
3. Save the `jazn-data.xml` file.

48.8.5 How to Grant the Application 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. Example 48–47 shows the code source grant where *<application-name>* is the application name specified in the `jazn-data.xml` policy store.

Example 48–47 Grant to Enable Policy Store Provisioning by Data Security Task Flow Source Code

```
<grant>
   <grantee>
      <codesource>
          <url>file:${domain.home}/servers/${weblogic.Name}/tmp/_WL_user/<application-name>/-</url>
      </codesource>
   </grantee>
   <permissions>
      <permission>
          <class>oracle.security.jps.service.policystore.PolicyStoreAccessPermission</class>
          <name>context=APPLICATION,name=<application-name></name>
          <actions>getApplicationPolicy,grant,revoke,alterAppRole,createAppRole,
                                                    alterAppRoleCategory</actions>
      </permission>
   </permissions>
</grant>
```

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 Example 48–47 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.
3. Save the `jazn-data.xml` file.

48.8.6 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. Once deployed, the application will make use of the application roles and security policies defined by the application stripe. Example 48–48 shows the `web.xml` entry to identify the existing application stripe.

Example 48–48 web.xml Parameter Identifies Deployed Application Stripe

```
<init-param>
      <param-name>application.name</param-name>
      <param-value><application-name></param-value>
</init-param>
```

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 Example 48–48 beneath the JPS filter.

   The application name you enter must match the application name identified in the policy store definition.
3. Save the `web.xml` file.

49

Implementing Function Security

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:

49.1 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 the web pages of 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 Chapter 48, "Implementing Oracle Fusion Data Security."

Function security controls access to securable application artifacts including ADF task flows and top-level web pages backed by ADF page definition files. Users who do not have the required privilege cannot view the task flow. 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 not enter the task flow.

49.1.1 Function Security Development Environment

Before you can implement function security for custom application resources, an IT security manager 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.

Important: 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. You should always add custom application roles to grant access rights. For details about restrictions when working with the file-based policy store, see Section 49.3.7, "What You May Need to Know About Actions That Developers Must Not Perform."

As an security implementation guideline, 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 create for Oracle ADF application artifacts.

After you customize security, you use JDeveloper to add end user identities to the identity store of the exported file for the purpose of 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 IT security manager 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.

49.1.2 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.

To gather background information about function security, refer to these documents:

- *Oracle Fusion Applications Extensibility Guide*

  The main document addressing how to customize and extend Oracle Fusion applications. For details 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 Business Objects and Application Artifacts" chapter.

- *Oracle Fusion Applications Security Guide*

  The main document addressing the concepts and best practices of the Oracle Fusion security approach.

- *Oracle Fusion Middleware Application Security Guide*

  The main document addressing the concepts and best practices of Oracle Platform Security Services (OPSS) upon which Oracle Fusion security is based.

- *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*

  Describes ADF Security, through which ADF components interact with OPSS.

Table 49–1 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 49–1 Oracle Fusion Function Security Use Cases

Security Goal	Security Policy Requirement	Application Developer Tasks
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.	Create a new entitlement grant. The new task flow will be inaccessible by default (also called *protected*) 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.	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 *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 an IT security manager to create for you. For more information, see Section 49.3.1, "How to Create Entitlement Grants for Custom Application Roles."
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.	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.	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 an IT security manager to create for you. For more information, see Section 49.3.1, "How to Create Entitlement Grants for Custom Application Roles."
Control whether a **new task flow** or a **new top-level web page** is publicly accessible. *Publicly accessible* means the application resource may accessed by guest users (those who do not need to log into the application) or it can mean to all authenticated users who are not provisioned with the privileges conferred by a custom application role.	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.	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 Section 49.3.3, "How to Define Resource Grants for OPSS Built-In Roles."

Table 49–1 (Cont.) Oracle Fusion Function Security Use Cases

Security Goal	Security Policy Requirement	Application Developer Tasks
Determine whether the end user associated with a particular role has the right to select **create, edit, or delete buttons** in the displayed web page.	Do not create a security policy. Access to user interface components, such as buttons, is not controlled by a security policy, but can be controlled by rendering the button in the user interface based on the end user's role.	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 membership in a particular role. For more information about rendering components using EL utility methods, see the "Enabling ADF Security in a Fusion Web Application" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

49.1.3 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.

Table 49–2 lists the file related to Oracle Fusion security that are created for you when you secure your application in JDeveloper. For more information about the security-related files, see the *Oracle Fusion Middleware Application Security Guide*.

Table 49–2 Oracle Fusion Application Security-Related Files

File Name	Location	Security Purpose
`jazn-data.xml`	`<JDevAppHome>/src/META-INF`	■ Design-time policy store, containing all the function security policies defined for securable Oracle ADF artifacts in the workspace. This file must be generated by an IT security manager.
`adf-config.xml`	`<JDevAppHome>/.adf/META-INF`	■ Stores the flags that control whether authentication and authorization are supposed to be enforced by runtime Oracle ADF. ■ Specifies the security providers' and OPSS factory's classes names.
`weblogic-application.xml`	`<JDevAppHome>/src/META-INF`	■ Defines the listeners used by Oracle WebLogic Server to interact with the application. ■ Controls the application stripe to which policies are migrated.
`jps-config.xml`	`<JDevAppHome>/src/META-INF`	■ Stores definitions of OPSS service providers and services instances. ■ Stores definitions of OPSS contexts for the current workspace. An OPSS context contains pointers to the service instances used by that application.

Table 49–2 (Cont.) Oracle Fusion Application Security-Related Files

File Name	Location	Security Purpose
`web.xml`	`<JDevAppHome>/UserInterface/public-html/WEB-INF`	■ Stores definitions and mappings for servlets and filters used by ADF Security. ■ Includes security constraints for any custom servlets. ■ Defines the OPPS `JpsFilter` servlet filter to set up the OPSS policy provider. The filter defines settings that indicate that your servlet has special privileges. It is important that the `JpsFilter` definition be the first filter definition in the `web.xml` file. ■ Adds the ADF `adfAuthentication` servlet definition to require the user to log in the first time the application is accessed. ■ Maps the `adfAuthentication` servlet to a security constraint that triggers user authentication dynamically. ■ Defines the `ADFBindingFilter` servlet filter which instantiates the `ADFContext` object, which includes context information about the session, including the security context. ■ Defines login configuration for the application. ■ Define the Java EE logical `valid-users` role, which is used to trigger the security constraint that enables dynamic authentication. ■ Contains the runtime application stripe, which must match the configuration in the `weblogic-application.xml` file.
`weblogic.xml`	`<JDevAppHome>/UserInterface/public-html/WEB-INF`	■ Maps the Java EE `valid-users` role (created by the Configure ADF Security wizard) to the implicit `users` group defined by Oracle WebLogic Server.
`cwallet.sso`	`<JDevAppHome>/src/META-INF`	■ Stores external system's password in encrypted format.

49.2 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 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.

To implement function security:

1. Decide the names of custom application roles that your application will specify as the grantee of security privileges.

   For information about application roles, see the "Understanding Security Concepts" part of the *Oracle Fusion Middleware Application Security Guide*.

2. Optionally, ask the IT security manager to create custom application roles with the names you supply.

   If a security manager 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. For details about how the IT security manager creates application roles using Oracle Authorization Policy Manager, see the "Managing Security Artifacts" chapter in the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

   If you do not ask the security manager to create custom application roles, then you must create them in JDeveloper before you define security policies.

3. Consult an IT security manager to export all predefined function security policies of the application that you are customizing into a `jazn-data.xml` file.

   For details about how the security manager exports the application policy store, see the "Securing Oracle Fusion Applications" chapter in the *Oracle Fusion Applications Administrator's Guide*.

4. 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.

5. 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 Section 49.3.1, "How to Create Entitlement Grants for Custom Application Roles."

6. 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 Section 49.3.3, "How to Define Resource Grants for OPSS Built-In Roles."

7. 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 Section 49.3.5, "How to Enforce Authorization for Securable ADF Artifacts."

8. Determine which user interface components you want to associate with user entitlements, and enter 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). For details about using EL utility methods, see the "Enabling ADF Security in a Fusion Web Application" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

9. 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 Section 49.3.6, "How to Enable Authentication and Test the Application in JDeveloper."

10. After testing is complete, remove the test users from the `jazn-data.xml` file and provide the updated `jazn-data.xml` file to the IT security manager 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.

    For information about how the IT security manager merges the policies using Oracle Authorization Policy Manager, see the "Upgrading Oracle Fusion Applications Policies" chapter in the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

11. The IT security manager provisions enterprise users by mapping enterprise roles defined in the staging environment identity store to the custom application roles.

    For information about how the IT security manager provisions enterprise users using Oracle Authorization Policy Manager, see the "Managing Security Artifacts" chapter in the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

12. Before running the application in the staging environment, the IT security manager 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. For information about how the IT security manager reconciles GUIDs using a WLST command, see the "Securing Oracle Fusion Applications" chapter in the *Oracle Fusion Applications Administrator's Guide*.

13. 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 Section 49.3.8, "What You May Need to Know About Testing."

49.3 Adding 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.

> **Best Practice:** 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, the ADF artifacts that you may secure 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 Section 49.3.9, "What You May Need to Know About Security Best Practices."

To add function security to the application:

1. Determine which ADF artifacts should be secured and grant entitlement privileges to custom application roles to define the duties of end users.
2. Determine which ADF artifacts should be public and grant resource privileges to an appropriate OPSS built-in application role.
3. Opt into the previously defined function security policies by running the Configure ADF Security wizard to enforce OPSS authorization checking.
4. Determine which user interface components you want to associate with user entitlements, and enter 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). For details about

using EL utility methods, see the "Enabling ADF Security in a Fusion Web Application" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

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.

49.3.1 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 should be used only to define publicly accessible application artifacts, as described in Section 49.3.3, "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.

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 Permission** 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.

Table 49–3 lists the resource types displayed in JDeveloper and identifies the associated resource and actions.

Table 49–3 Resource Types of Securable ADF Artifacts

Resource Type	Supports These Resources and Actions
`ADF Task Flow`	Defines personalize, customize, grant, edit, and view actions on ADF bounded task flows in a Fusion Web application.
`ADF Region`	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.
`ADF Entity Permission`	Not used by Oracle Fusion Applications. Data security is provided by Oracle Fusion Data Security, as described in Chapter 50, "Securing Web Services Use Cases."
`ADF Method Resource`	Defines execute, invoke, and view actions on ADF methods in a Fusion web application.
`Webservice Resource`	Defines invoke actions on Fusion Web services. For more details about securing Web services, see Section 50.5, "Authorizing the Web Service with Entitlement Grants."

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 have an understanding of ADF Security. For more information, see the "Understanding Users and Roles" chapter in the *Oracle Fusion Middleware Application Security Guide*.

It may also be helpful to understand the details of the ADF Security containment model. For more information, see Section 49.3.9, "What You May Need to Know About Security Best Practices."

You will need to complete these tasks:

- Consult the IT security manager 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 Section 49.2, "Function Security Implementation Process Overview."
- Create application roles as described in the "Enabling ADF Security in a Fusion Web Application" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

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.

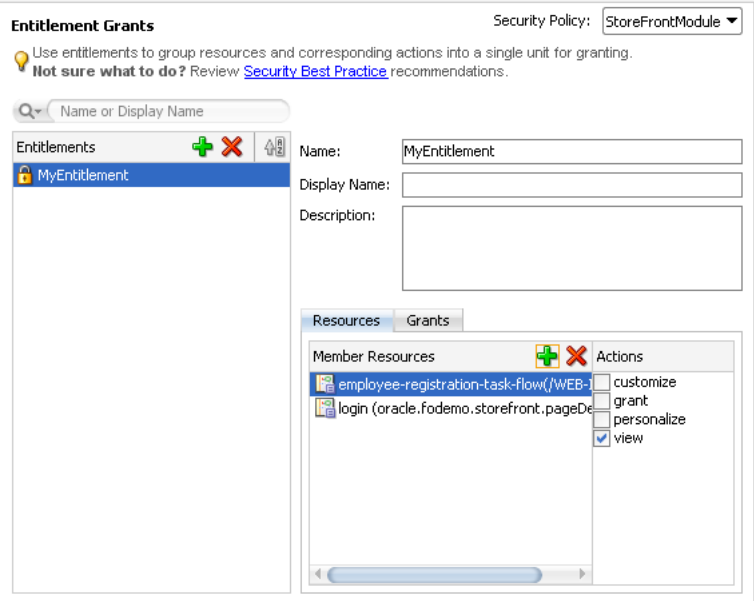
   The overview editor displays all the resources that your application defines.
3. In the Resources section, click the **Add Member Resource** icon to add a member resource to the entitlement.
4. In the Select Resources dialog, select the resource from the **Resource Type** dropdown and then select the 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 define by your selected project.
6. In the **Actions** lists, select the desired action for the selected resource.

   Figure 49–1 shows the overview editor with the **View** action selected for the task flow and added to **MyEntitlement**.

Figure 49–1 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 an IT security manager 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.

49.3.2 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 Example 49–1 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 49–1 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>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>
          <description>MyResource description</description>
          <type-name-ref>APredefinedResourceType</type-name-ref>
        </resource>
      </resources>

      <!-- entitlement definition -->
      <permission-sets>
        <permission-set>
          <name>MyEntitlement</name>
          <display-name>MyEntitlement display name</display-name>
          <description>MyEntitlement description</description>
          <member-resources>
            <member-resource>
              <type-name-ref>APredefinedResourceType</type-name-ref>
              <resource-name>MyResource</resource-name>
              <actions>write</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>MyEntitlement</name>
            </permission-set-ref>
          </permission-set-refs>
        </grant>
      </jazn-policy>
    </application>
  </applications>
 </policy-store>
</jazn-data>
```

49.3.3 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 in the *Oracle Fusion Middleware Application Security Guide*.

It may also be helpful to understand the details of the ADF Security containment model. For more information, see Section 49.3.9, "What You May Need to Know About Security Best Practices."

You will need to complete this task:

- Consult the IT security manager 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 Section 49.2, "Function Security Implementation Process Overview."

To grant public access to securable ADF artifact:

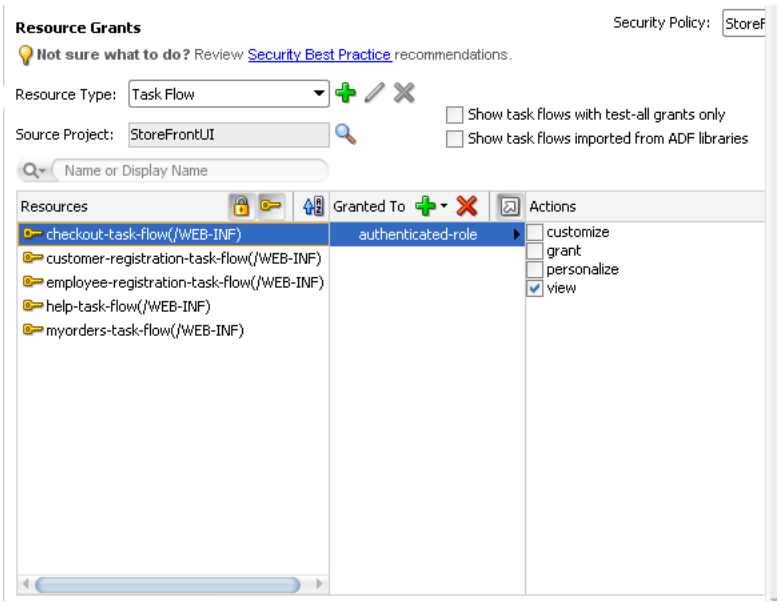
1. In the main menu, choose **Application** and then **Secure > Resource Grants**.
2. In the Resource Grants page of security policy overview editor, select the resource from the **Resource Type** dropdown and then select the desired project in the **Source Projects** list.

   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.

   Figure 49–2 shows the overview editor with the **View** action selected for the task flow and granted to **authenticated-role**.

Figure 49–2 Granting to authenticated-role in the Overview Editor



49.3.4 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.

Example 49–2 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 49–2 Resource-Based Policy Definition in the jazn-data.xml File

```
<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>
                  <class>oracle.adf.controller.security.TaskFlowPermission</class>
                  <name>/WEB-INF/customer-registration-task-flow.xml#
                                        customer-registration-task-flow</name>
                  <actions>view</actions>
            </permission>
            ...
         </permissions>
      ...
```

```
      </grant>
      ...
   </jazn-policy>
</policy-store>
```

49.3.5 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.

Once 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 in order 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 Section 49.3.1, "How to Create Entitlement Grants for Custom Application Roles." Alternatively, when you want to make a page pubic and accessible by unauthenticated user, you must explicitly grant to a built-in OPSS role, as described in Section 49.3.3, "How to Define Resource Grants for OPSS Built-In Roles."

Before you begin:

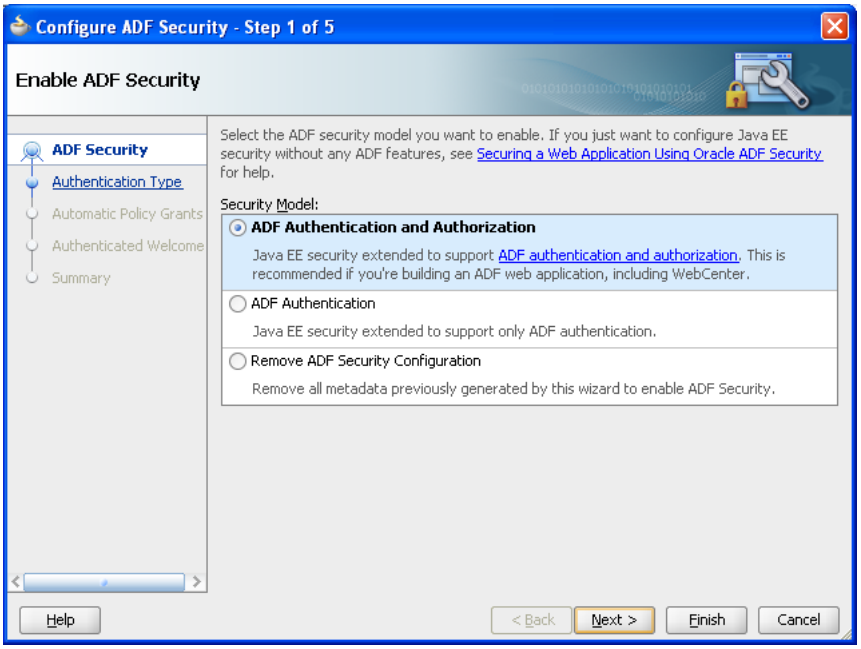
It may be helpful to have an understanding of the configuration changes made by the Configure ADF Security wizard. For more information, see the "Enabling ADF Security in a Fusion Web Application" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

To enforce authorization:

1. In the main menu, choose **Application** and then **Secure > Configure ADF Security**.
2. Update the wizard pages as follows:

   ADF Security: Select **ADF Authentication and Authorization** (default), as shown in Figure 49–3. Click **Next**.

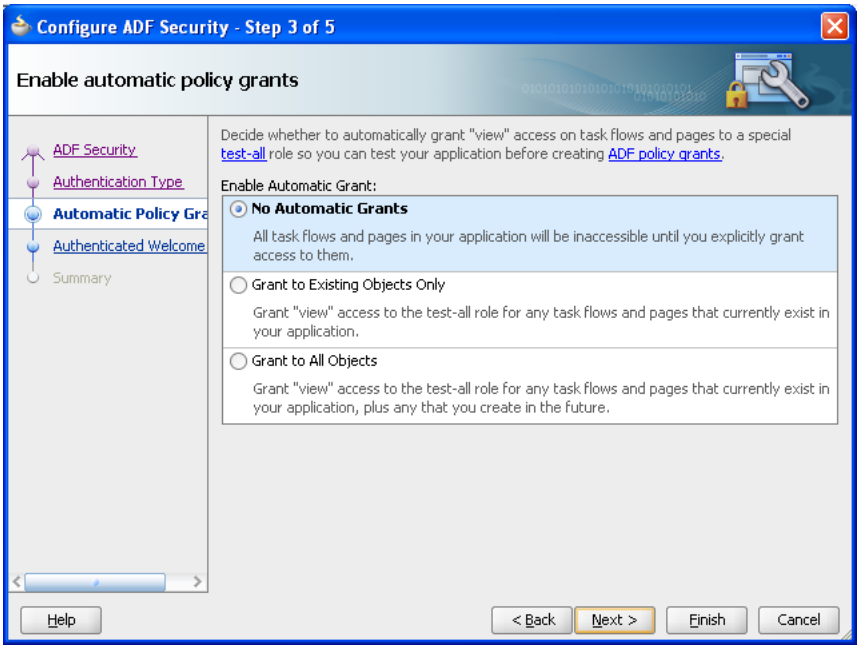
Figure 49–3 Configure ADF Security — ADF Security Model Page



Authentication Type: Select **HTTP Basic Authentication** (default). Click **Next**.

Automatic Policy Grant: Select **No Automatic Grants** (default), as shown in Figure 49–4. Click **Next**.

Figure 49–4 Configure ADF Security — Automatic Policy Grants Page



Note: When you select **No Automatic Grants**, you must define explicit grants that are specific to your application. The `test-all` application role provides a convenient way to run and test application resources without the restricted access that ADF authorization enforces. For more information about the `test-all` role, see the "Enabling ADF Security in a Fusion Web Application" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Authenticated Welcome: Do not make a selection. (The **Redirect Upon Successful Authentication** option should not be selected). Click **Next**.

Summary: Review your selections. Click **Finish**.

49.3.6 How to Enable Authentication and Test the Application in JDeveloper

Authentication is enabled when you run the Configure ADF Security wizard, as described in Section 49.3.5, "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).

For details about creating test users, see the "Enabling ADF Security in a Fusion Web Application" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

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.

49.3.7 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 *Oracle Fusion Applications Security Guide*.

49.3.8 What You May Need to Know About Testing

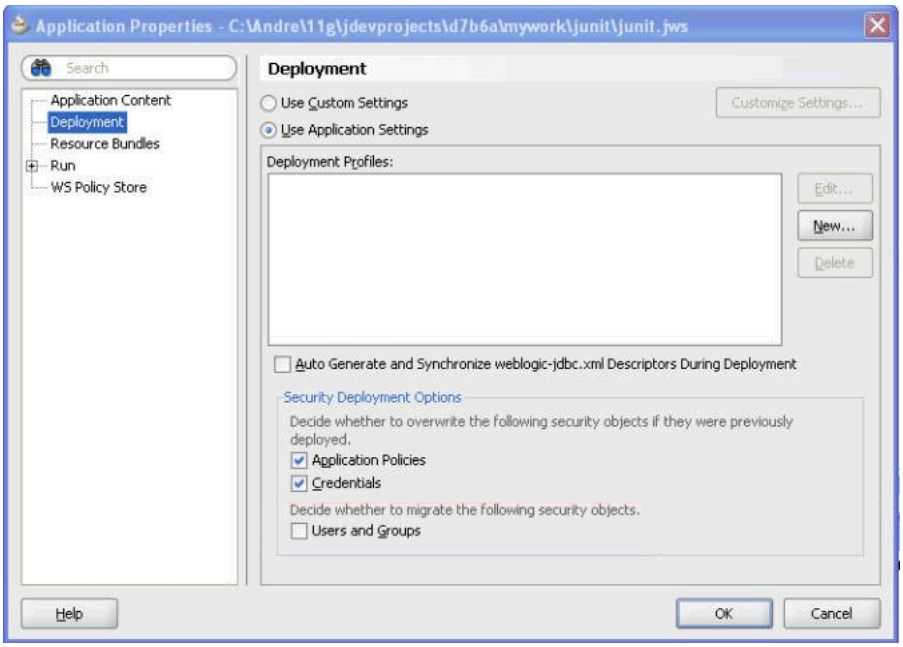
When the application needs to undergo testing either in your local environment or on a staging server, the following changes will ensure that the application uses the LDAP-based identity and policy stores configured on the staging server.

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.
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 Figure 49–5.

Figure 49–5 Application Deployment Properties Related to Security



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. For details about reconciling GUIDs in the data security repository, see the "Securing Oracle Fusion Applications" chapter in the *Oracle Fusion Applications Administrator's Guide*.

49.3.9 What You May Need to Know About Security Best Practices

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 once 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 single 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 by 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 once ADF Security is enabled. Therefore, it should never be deployed, since it grants access to the application for non-authenticated users.

50

Securing Web Services Use Cases

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:

50.1 Introduction to Securing Web Services Use Cases

The service use case patterns described in Part VI, "Common Service Use Cases and Design Patterns" must be secured. Security requirements vary depending on service and client implementations.

The following use case pattern represents an example of a typical pattern. The pattern includes some possible variations that require different security implementations. The use case example highlights the security implementation requirements of each service component.

An example service use case:

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 a Mediator component.

   Alternatively, you can trigger or call a SOA composite 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 (very rarely).
4. The Mediator component invokes a Business Process Execution Language (BPEL) process (unless the JAX-WS proxy or ADF Web service data control is used).

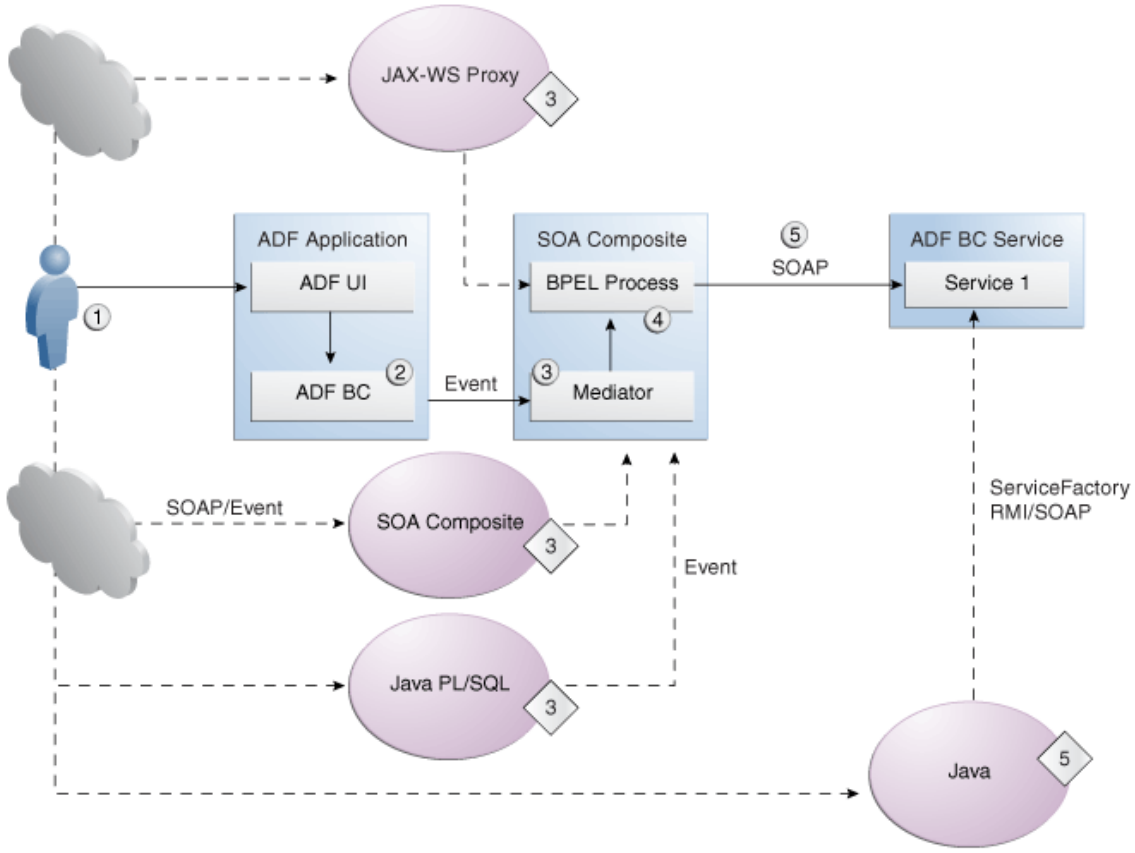
The Mediator can invoke various BPEL processes based on the incoming event. It can also transform the event to the payload that BPEL process takes.

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, it is also possible to generate events from another SOA composite, Java PL/SQL code, or Java code and either directly or indirectly invoke the ADF Business Components Web service. 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).

Figure 50–1 illustrates the possible event generation use cases for Oracle Fusion applications. The main use case flow—ADF Business Components-generated events—is illustrated in the center, along with numbers (enclosed in circles) illustrating the corresponding steps of the above use case. Possible alternative flows are represented by dashed lines and numbers (enclosed in boxes, again corresponding to the steps of the above use case).

Figure 50–1 Sample Web Service Use Case



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.

- 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.
- 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 an XML signature and encryption. Oracle WSM transparently handles this.
- Oracle WSM policies do not apply to events.

50.2 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. For complete details about how a system administrator attaches policies globally, see the "Understanding Oracle WSM Policy Framework" chapter in *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

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 needs to be invoked outside of an Oracle Fusion application that external service will be secured with message 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.

50.3 Attaching 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.

To enable global policy attachment:

1. Remove 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).

   For details about creating global policy sets, see the "Managing Web Service Policies" chapter in *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

2. Before enabling GPA, make sure that you have attached the `no_authentication_service_policy` policy to those services that do not need 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.

Developers can do this in Oracle JDeveloper directly in the ADF Business Components Web service implementation class file, as described in Section 50.4, "Attaching Policies Locally." Also, a system administrator can do this either using Oracle Enterprise Manager Fusion Middleware Control or using WebLogic Scripting Tool (WLST).

For details about attaching local policies on Oracle WebLogic Server, see the "Attaching Policies to Web Services" chapter in *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

3. Decide 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 50–1.

   For a summary of the predefined policies, see the "Predefined Policies" appendix in *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

4. Create 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.

   For details about creating global policy sets, see the "Creating and Managing Policy Sets" chapter in *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

5. Verify that your client and Web services are using GPA.

   A system administrator does this using either Oracle Enterprise Manager Fusion Middleware Control or using WLST.

   For details about creating global policy sets, see the "Creating and Managing Policy Sets" chapter in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

50.4 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
- 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

Table 50–1 shows the recommended Oracle WSM policies and the components to which they apply.

Table 50–1 Recommended Oracle Web Services Manager ***Policies for Oracle Fusion Applications***

Profile	Service Side Policy	Client Side Policy	Features
Authentication (AuthN)	`wss_saml_or_username_token_service_policy`	Username: `wss10_saml_token_client_policy` SAML: `wss10_saml_token_client_policy`	Performance: High Security: Low - Authentication: password in clear, SAML token is unsigned. - Wire level security: No - Hardening: no Configuration: Easy. No key stores to set up. Interoperability: - Username: High interoperates easily with many different stacks, including .NET, SOAP-UI, and more. - SAML: Low. Unsigned SAML SV does not interoperate with most stacks.
SSL Profile	`wss_saml_or_username_token_over_ssl_service_policy`	Username: `wss10_saml_token_client_policy` SAML: `wss10_saml_token_client_policy`	Performance: Medium Security: Medium - Authentication: passwords encrypted because of 1-way-SSL, SAML token is signed by virtue of 2-way-SSL. - Wire level security: Transport level, using 1-way-SSL for username, and 2-way-SSL for SAML. - Hardening: Medium. Each application server can have its own separate key. Configuration: Hard. The same OHS URI must be set up for both 1-way and 2-way-SSL and the client certificate needs to be set propagated from OHS to Oracle WLS. For details, see the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*. Interoperatiblity: - Username: High - SAML: Medium
Message Security	`wss11_saml_or_username_token_with_message_protection_service_policy`	Username: `wss10_saml_token_client_policy` SAML: `wss10_saml_token_client_policy`	Performance: Medium Security: High - Wire level security: Transport level - Hardening: High. Not only each server, but each Web service can have its own separate key. Configuration: Medium. Key stores need to be set up. Interoperatiblity: - Username: Medium - SAML: Medium

Table 50–1 (Cont.) Recommended Oracle Web Services Manager *Policies for Oracle Fusion*

Profile	Service Side Policy	Client Side Policy	Features
No Behavior	`oracle/no_authentication_service_policy`	`oracle/no_authentication_client_policy`	No security. Service will be accessible.

50.4.1 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. Once 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.

For more information about directly attaching the no behavior policy to a Web service endpoint, see the "Creating and Managing Policy Sets" chapter in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

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 Example 50–1.

Example 50–1 Enabling No Behavior Policy for an ADF Business Components Web Service

```
@SecurityPolicy( { "oracle/no_authentication_service_policy"})
@CallbackSecurityPolicy("oracle/no_authentication_service_policy")
```

For more information about ADF Business Components Web services, see the "Integrating Service-Enabled Application Modules" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

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 in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

50.4.2 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 Example 50–2

Example 50–2 Attaching a Local Policy for an ADF Business Components Web Service

```
@SecurityPolicy( { "oracle/wss11_saml_or_username_token_with_message_protection_
service_policy"})
@CallbackSecurityPolicy("oracle/wss11_saml_token_with_message_protection_client_
policy")
```

Note that Fusion Web services should have asynchronous method calls enabled.

50.4.3 How to Provide Additional Security Hardening for Web Service Clients

Typically, Fusion Web services use a common domain wide key, which should be used 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.

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 *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

50.4.4 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.

For details about locally attaching Web service client policy on Oracle WebLogic Server, see the "Attaching Policies to Web Services" chapter in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

50.5 Authorizing 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 an IT security manager to export all predefined function security policies of the application that you are customizing into a `jazn-data.xml` file.

   For details about how the security manager exports the application policy store, see the "Securing Oracle Fusion Applications" chapter in the *Oracle Fusion Applications Administrator's Guide*.

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.
2. Enforce ADF Security authorization for the Web service in the Web service implementation class.

50.5.1 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 polices 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. You should use custom duty roles and you should not grant entitlements to predefined duty roles.

For details about creating entitlement-based security policies using JDeveloper tools, see Section 49.3.1, "How to Create Entitlement Grants for Custom Application Roles."

Example 50–3 shows a complete set of required grants enabling Web service authorization.

Example 50–3 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>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>WebserviceResourceType</name>
          <display-name>WebserviceResourceType</display-name>
          <description>Webservice Resource</description>
          <provider-name />
          <matcher-class>oracle.wsm.security.WSFunctionPermission</matcher-class>
          <actions-delimiter>,</actions-delimiter>
          <actions>invoke</actions>
        </resource-type>
      </resource-types>

      <resources>
        <resource>
          <name>http://xmlns.oracle.com/apps/financials/subledgerAccounting
                  /accountingMethodSetup/accountRulesService/AccountRulesService#*
          </name>
          <type-name-ref>WebserviceResourceType</type-name-ref>
        </resource>
        <resource>
          <name>http://xmlns.oracle.com/apps/contracts/termsAuthoring/deliverables
               /service/DeliverableService#findDeliverableByDeliverableId
          </name>
          <type-name-ref>WebserviceResourceType</type-name-ref>
        </resource>
      </resources>

      <!-- entitlement definition -->
      <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>
  </policy-store>
</jazn-data>
```

For details about the ADF Security, see the "Enabling ADF Security in a Fusion Web Application" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

50.5.2 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.

Table 50–2 shows the recommended policy interceptor used to enforce entitlement-based policies for Oracle Fusion applications and the components to which they apply.

Table 50–2 Recommended ADF Business Components Policy Interceptor

On this component...	Use this interceptor...
ADF Business Components Web services (at the service or operation level)	Use `ServicePermissionCheckInterceptor` to implement authorization.

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, as long as `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 Example 50–4.

Example 50–4 Enforcing Authorization with ADF Business Components Policy Interceptor

```
import oracle.jbo.server.svc.ServicePermissionCheckInterceptor;
import oracle.jbo.server.svc.ServiceContextInterceptor;
@Interceptors({ServiceContextInterceptor.class, ServicePermissionCheckInterceptor.class})
public class xxxServiceImpl ...
```

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 Example 50–5.

Example 50–5 Configuring the JPSInterceptor for the Application in the ejb-jar.xml File

```
<?xml version = '1.0' encoding = 'windows-1252'?>
<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"
         xmlns="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>
```

For details about the ADF Business Components `ServiceFactory` class, see the "Integrating Service-Enabled Application Modules" chapter in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Development Framework*.

50.6 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).

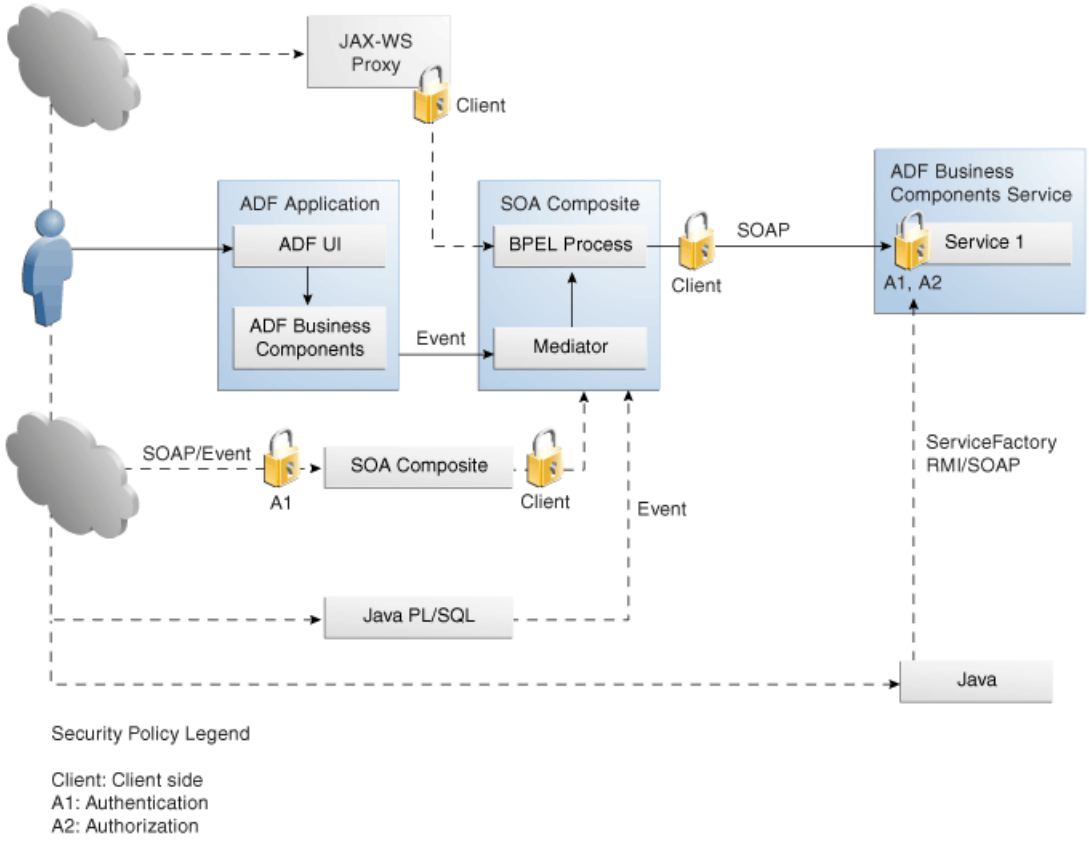
The client side can invoke the service through a SOAP service invocation, where the client can be a JAX-WS proxy or a SOA composite (service factory via SOAP is still JAX-WS proxy). Before passing the service to the server side, the client side will either propagate the current user credential or switch 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.

Figure 50–2 illustrates how Oracle WSM policies are enforced within the Oracle Fusion Applications use case described in Section 50.1, "Introduction to Securing Web Services Use Cases."

Figure 50–2 Using Oracle Web Services Manager Security Policies



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.

For more information about Oracle Web Services Manager policies, see the "Understanding Oracle WSM Policy Framework" chapter in *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

51

Securing 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.

51.1 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`) in order to override GPA.

The requirements for the client counterparts for each of these ports is exactly 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.

Table 51–1 summarizes the policies attached to the portlet service and the client.

Table 51–1 Recommended Oracle Web Services Manager Policies for Oracle Fusion Portlets

Port	Service Side Policy	Client Side Policy
`WSRP_v2_Markup_Service`	No local policy. Inherits from GPA, which by default is `oracle/wss_saml_or_username_token_service_policy`.	No local policy. Inherits from GPA, which by default is `oracle/wss_saml_or_username_token_client_policy`.
`WSRP_v2_PortletManagement_Service`	Local anonymous policy: `oracle/no_authentication_service_policy`	Local anonymous policy: `oracle/no_authentication_client_policy`
`WSRP_v2_Registration_Service`	Local anonymous policy: `oracle/no_authentication_service_policy`	Local anonymous policy: `oracle/no_authentication_client_policy`
`WSRP_v2_ServiceDescription_Service`	Local anonymous policy: `oracle/no_authentication_service_policy`	Local anonymous policy: `oracle/no_authentication_client_policy`

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.

51.2 Securing 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

51.2.1 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 Example 51–1.

Example 51–1 Edit the oracle-webservices.xml File

```
<port-component name="WSRP_v2_Markup_Service" style="document"
bindingQName="{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>
   <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"/>
   </operations>
   <!-- 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:wsdl" localpart="WSRP_v2_Service"/>
     <soap-version>soap1.1</soap-version>
   </deployment>
   <!-- end:deployment time generated info   -->
     <servlet-link>WSRP_v2_Markup_Service</servlet-link>
</port-component>
```

51.2.2 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 Example 51–2. This code should be configured out of the box.

Example 51–2 Verify the credstore and keystore serviceInstance Elements

```
<serviceInstance location="./" provider="credstoressp" name="credstore">
            <description>File Based Credential Store Service Instance</description>
</serviceInstance>

<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>
```

3. Make sure the default context references the key store and credential store service instances as shown in Example 51–3.

Example 51–3 Default Context References to the Credential Store and Key Store

```
<jpsContext name="default">
            <serviceInstanceRef ref="credstore"/>
            <serviceInstanceRef ref="policystore.xml"/>
            <serviceInstanceRef ref="audit"/>
            <serviceInstanceRef ref="idstore.ldap"/>
            <serviceInstanceRef ref="keystore"/>
</jpsContext>
```

Note: There is no need to restart your domain if this configuration is already in place.

51.2.3 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.

Once 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 Example 51–4.

Example 51–4 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 51–5 shows an entitlement grant enabling access to the task flow.

Example 51–5 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>F5494E409CFB11DEBFEBC11296284F58</guid>
          <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</type-name-ref>
            <type-name-ref>TaskFlowResourceType</type-name-ref>
            <display-name>my-task-flow</resource-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>
          <principals>
            <principal>
              <class>
                 oracle.security.jps.service.policystore.ApplicationRole
              </class>
              <name>AppRole</name>
            </principal>
          </principals>
        </grantee>

        <!-- entitlement granted to an application role -->
        <permission-set-refs>
          <permission-set-ref>
            <name>MyPortletEntitlement</name>
          </permission-set-ref>
        </permission-set-refs>
      </grant>
    </jazn-policy>
  </application>
 </applications>
</policy-store>
</jazn-data>
```

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 Section 51.3, "Securing the Portlet Client."

51.3 Securing 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 should be used 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 Section 51.2.2, "How to Configure the Key Store and Credential Store."

51.4 Registering 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.

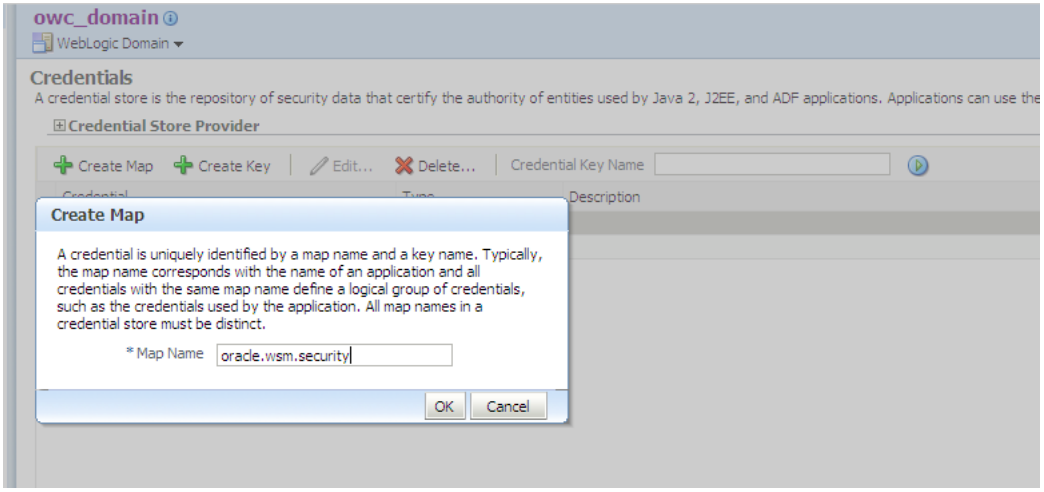
51.4.1 How to Register the Key Store and Write to the Credential Store

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. Figure 51–1 displays the Create Map window.

Figure 51–1 Create a Map If None Exists

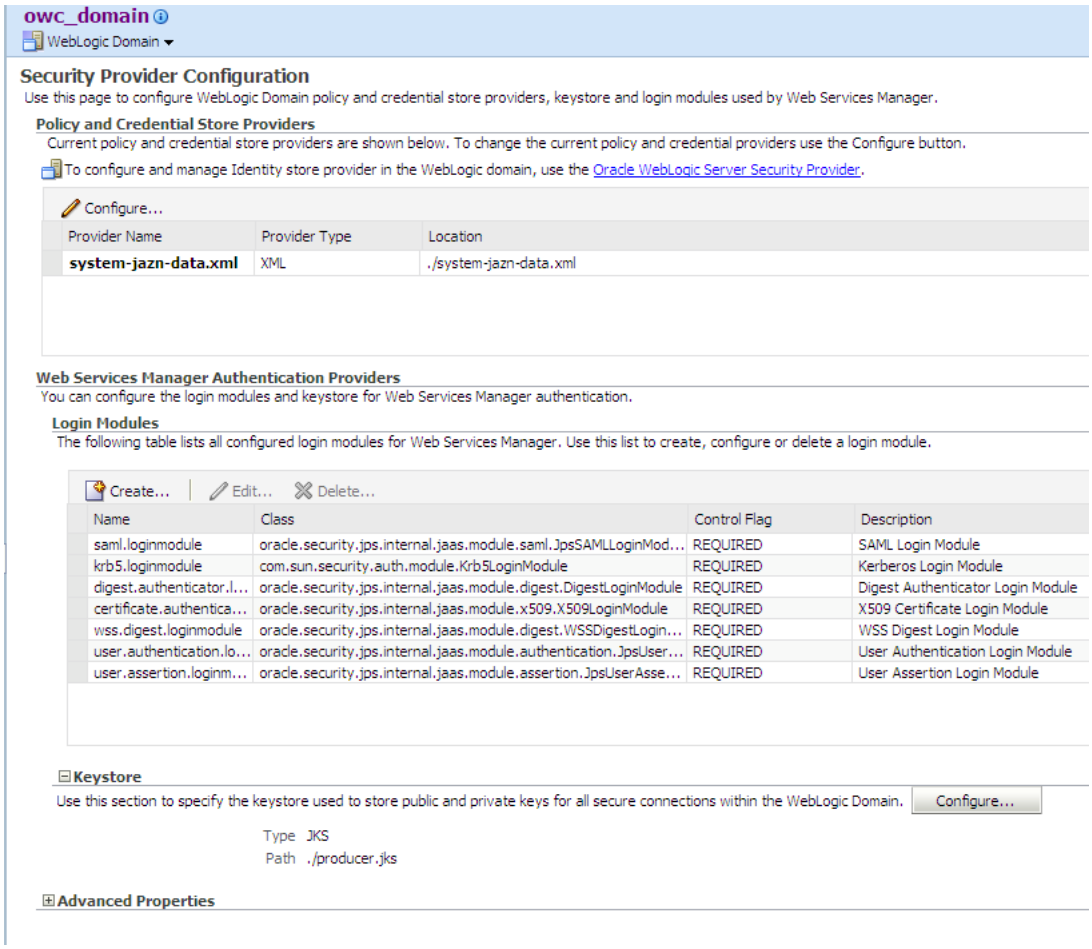


   - Click the **Create Map** button.
   - For the **Map Name**, enter `oracle.wsm.security`.

     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 Figure 51–2.

Figure 51–2 Click the Configure Button

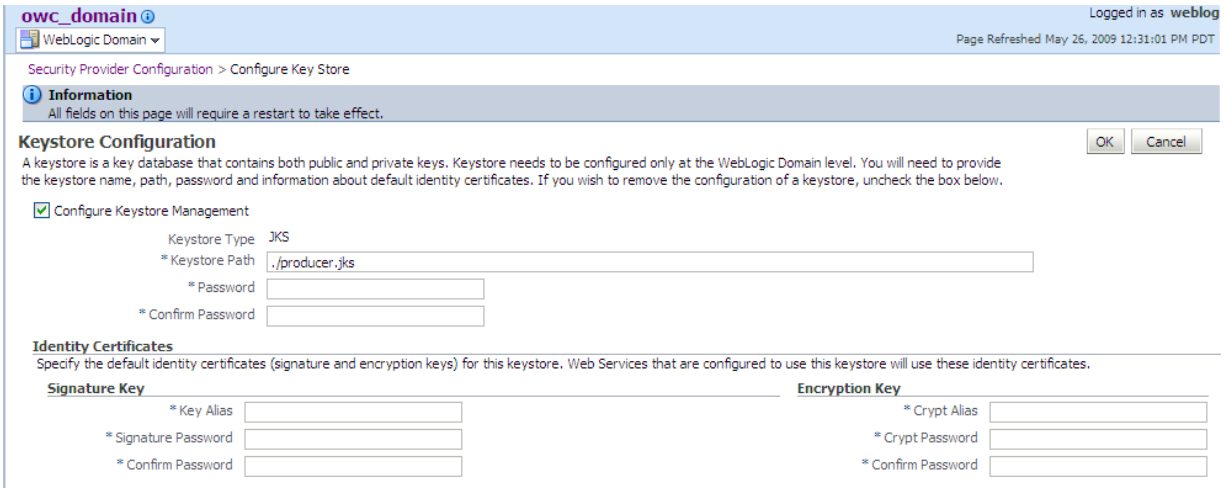


6. If the Keystore Path displays `./default-keystore.jks`, follow the instructions here. Otherwise, skip this step.

   Uncheck the **Configure KeyStore Management** box and click **OK**. This displays the window shown in Figure 51–2.

   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`. Figure 51–3 displays the Keystore Configuration page.

Figure 51–3 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.

51.4.2 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.

Part VIII

Advanced Topics

This part of the Developer's Guide provides information about some of the advanced features that are part of Oracle Fusion. These advanced features include the Oracle WebLogic Server, repositories used in Oracle Fusion, profiles, Oracle Fusion application seed data, and the Oracle Fusion Database Schema Deployment Framework. Also included in this part, are procedures for debugging Oracle Application Development Framework (Oracle ADF) and service-oriented architecture (SOA) applications.

Oracle WebLogic Server: Deployment is the process of packaging application files as an archive file and transferring it to a target application server. You can use JDeveloper to deploy your ADF applications directly to the WebLogic Server or indirectly to an archive file as the deployment target. You can then install this archive file to the target server. You can also run applications in JDeveloper using the Integrated WebLogic Server.

The *Creating Repository Connections* chapter provides information about the repositories that are used in Oracle Fusion and describes how to connect to each of these repositories using JDeveloper. The repositories include Oracle Universal Content Management, the Oracle Data Integrator (ODI), and Oracle Business Activity Monitoring (Oracle BAM).

A *profile* is a set of changeable options that affect the way your application looks and behaves. Profiles control how applications operate for users by the values that are set. Profiles can be set at different levels depending on how the profiles are defined.

Oracle Fusion Middleware Application Seed Data is the essential data to enable Oracle Fusion Middleware applications. Some examples include static lists of values, functional or error messages, and lookup values. The Seed Data Utility, which runs under JDeveloper, provides data extraction from the development instances of Oracle Fusion Applications. It also loads the extracted data to the customer database instances of Oracle Fusion Applications by integrating with Oracle ADF TaskManager. This part discusses how to set up your seed data environment, and how to extract and upload seed data.

The *Using the Oracle Fusion Database Schema Deployment Framework* (applxdf) includes JDeveloper plugins that handle applications-specific metadata, datamodeling 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 like Task Director. The Database Schema Deployment Framework is packaged and delivered to Oracle Fusion Applications and technology teams.

The *Improving Performance* chapter contains performance, scalability, and reliability (PSR) best practices documented based on performance analysis of several

prototypical Oracle Fusion Applications as well as various tests conducted by the Oracle Fusion middleware performance team. The outcome of this analysis is captured in this chapter. It includes best practices for coding and tuning ADF Business Components-based applications with performance, scalability, and reliability in mind.

The *Debugging Oracle ADF and Oracle SOA Suite* 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.

Designing and Securing View Objects for Oracle Business Intelligence Applications provides guidelines and best practices for designing and securing view objects and other supporting ADF Business Components objects for use by Oracle Fusion Business Intelligence (BI) Applications.

Implementing ADF Desktop Integration describes how Oracle Application Development Framework 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.

The Oracle Metadata Services (MDS) framework allows you to create customizable applications. The *Creating Customizable Applications* chapter describes how to configure your application at design time so that it can be customized. It also provides information about how to customize your applications using JDeveloper and WebCenter Composer.

This part contains the following chapters:

- Chapter 52, "Running and Deploying Applications on Oracle WebLogic Server"
- Chapter 53, "Creating Repository Connections"
- Chapter 54, "Defining Profiles"
- Chapter 55, "Initializing Oracle Fusion Application Data Using the Seed Data Loader"
- Chapter 56, "Using the Database Schema Deployment Framework"
- Chapter 57, "Improving Performance"
- Chapter 58, "Debugging Oracle ADF and Oracle SOA Suite"
- Chapter 59, "Designing and Securing View Objects for Oracle Business Intelligence Applications"
- Chapter 60, "Implementing ADF Desktop Integration"
- Chapter 61, "Creating Customizable Applications"

52

Running and Deploying Applications on Oracle WebLogic Server

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 for the purpose of performing end-to end testing of new applications. If you are deploying customizations or extensions, see the *Oracle Fusion Applications Extensibility Guide*.

This chapter includes the following sections:

- Section 52.1, "Introduction to Deploying Applications to Oracle WebLogic Server"
- Section 52.2, "Running Applications on Integrated WebLogic Server"
- Section 52.3, "Preparing to Deploy Oracle ADF Applications to an Administration Server Instance of WebLogic Server"
- Section 52.4, "Deploying Your Oracle ADF Applications to an Administration Server Instance of WebLogic Server"
- Section 52.5, "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.

52.1 Introduction to Deploying Applications to Oracle WebLogic Server

Deployment is the process of packaging application files as an archive file and transferring 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.

If you are using Integrated WebLogic Server, JDeveloper already provides the environment to run the application using the **Run** command. You do not need to create deployment descriptors or create standalone WebLogic Server domains. For more information on using Integrated WebLogic Server, see Section 52.2, "Running Applications on Integrated WebLogic Server."

If you are deploying the application to a standalone WebLogic Server instance, you must perform several tasks 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 Chapter 2, "Setting Up Your Development Environment."

Table 52–1 describes some common deployment techniques that you can use during the application development and deployment cycle.

Table 52–1 Deployment Techniques for Development Environments

Deployment Technique	Environment	When to Use
Run directly from JDeveloper	Test or Development	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.
Use JDeveloper to directly deploy to the target application server	Test or Development	When you are ready to deploy and test your application on an application server in a test environment. For example, you can also use the test environment to develop your deployment scripts.
Use JDeveloper to deploy to an EAR file, then use the Oracle WebLogic Server Administration console, WLST commands, or Enterprise Manager for deployment.	Test or Development	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. You can also use the test environment to develop your deployment scripts.

All WebLogic Server instances within the same domain must be at the same major and minor version. Servers within a domain can be at different maintenance pack levels as long as the Administration Server Weblogic Server instance is at the same maintenance pack level or higher than WebLogic Server instances called Managed Servers. For more information about the tasks that are required, see Section 52.3, "Preparing to Deploy Oracle ADF Applications to an Administration Server Instance of WebLogic Server."

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 Section 52.4, "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 Administrator's Guide* and the *Oracle Fusion Middleware Administrator's Guide for Oracle Application Development Framework*.

Note: This chapter discusses deploying applications. If you are deploying customizations or extension, see the "Deploying ADF Customizations and Extensions" and the "Deploying SOA Composite Customizations and Extensions" sections of the *Oracle Fusion Applications Extensibility Guide*.

52.1.1 Prerequisites for Deployment

Before you deploy an application, you should perform the following tasks:

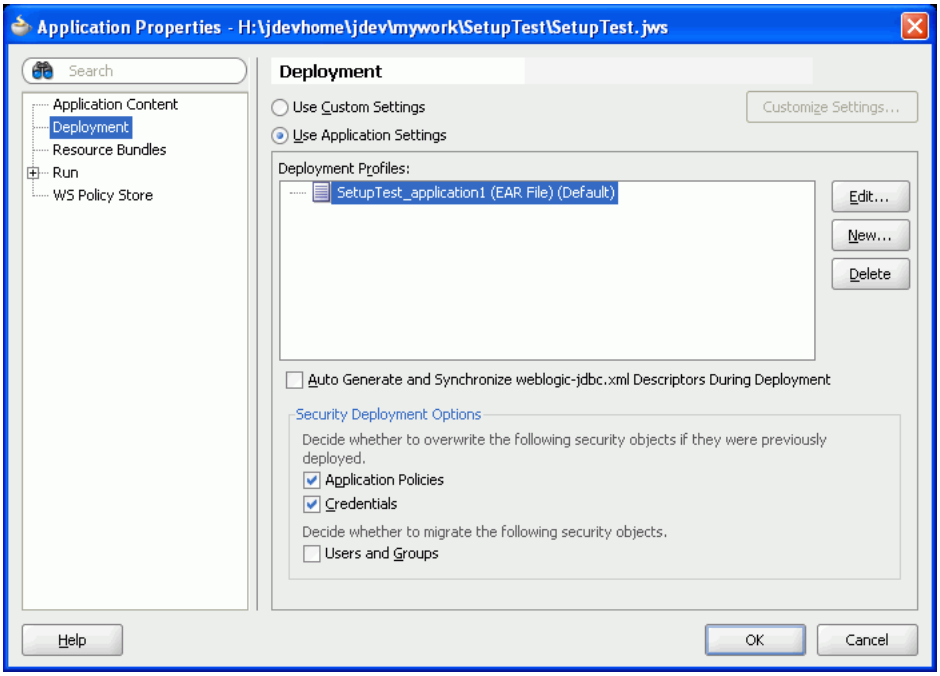
1. You must run the Configure Fusion Domain Wizard to set up the Integrated WebLogic Server domain or create a property file to be used to set up the standalone WebLogic Server domain. For instructions, see Chapter 2, "Setting Up Your Development Environment."
2. If you want more debugging output, set the environment variable using one of the shell commands as shown in Example 52–1, before starting the Administration Server instance of WebLogic Server or before starting JDeveloper if you are going to run Integrated WebLogic Server.

Example 52–1 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"
```

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 52–1. Unselect the **Auto Generate and Synchronize weblogic-jdbc.xml Descriptors During Deployment** option and click **OK**.

Figure 52–1 Application Properties — Deployment Dialog



52.1.2 Introduction to the Standalone Administration Server WebLogic Server Instance

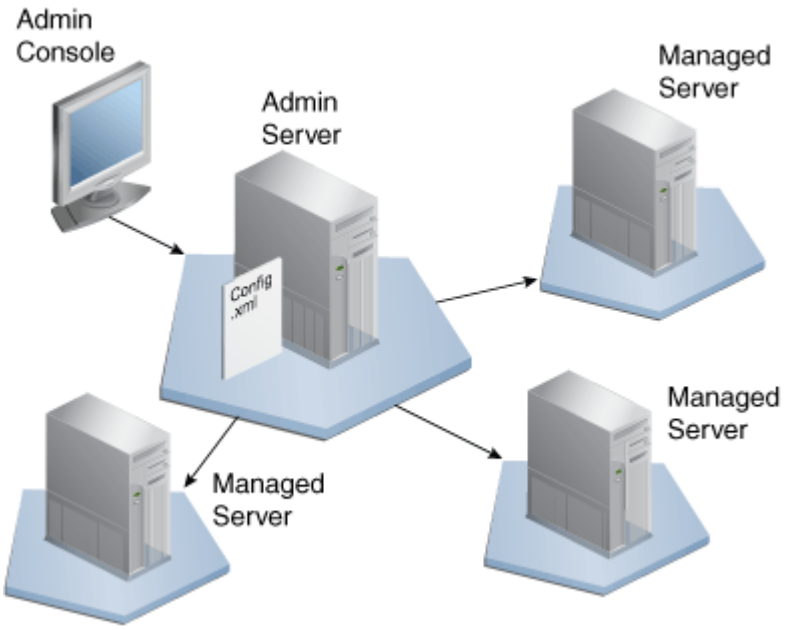
A Weblogic Server instance is a configured instance to host applications, such as Web applications, Enterprise applications, and Web services, and resources, such as Java Message Service (JMS), and JDBC, Diagnostics.

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.

Figure 52–2 Administration Server Configuration



There is only one Administration Server WebLogic Server instance in a domain, and an Administration Server WebLogic Server instance 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 WebLogic Server instance 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 WebLogic Server instance 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 WebLogic Server instance and as the host for applications, as illustrated in Figure 52–3.

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

Figure 52–3 Single Server Environment

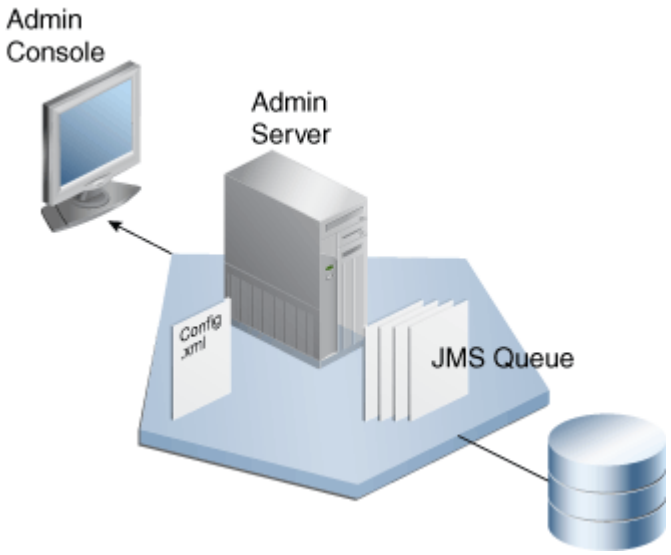
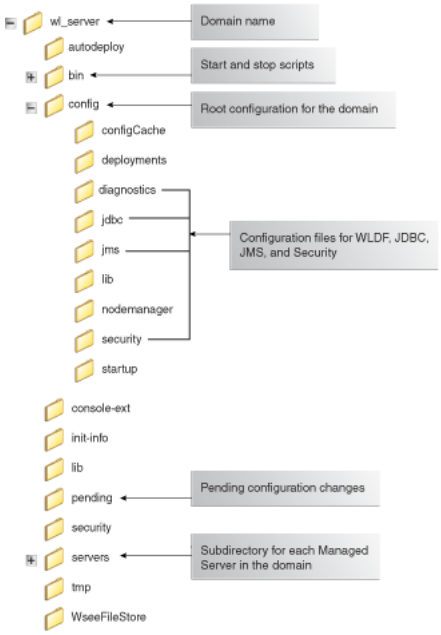


Figure 52–4 illustrates the structure of the domain directory:

Figure 52–4 Domain Directory Structure



You deploy applications to the Administration Server WebLogic Server instance. Only Administration Servers and Managed Servers are used in a production environment. Therefore, all your end-to-end testing should be done using the Administration Server.

There are two types of 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 SDO co-location requirement and need to be deployed to the SOA container. If the service must be deployed to the SOA-configured WebLogic Server, create a new EAR profile containing only that service or services from your application workspace.

For information about how to configure the SOA WebLogic Server, see Section 2.3, "Setting Up the Personal Environment for Standalone WebLogic Server."

52.2 Running Applications on Integrated WebLogic Server

Integrated WebLogic Server is a single server that is included within JDeveloper. You can run your applications directly on this server without needing to deploy. 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. However, real end-to-end testing should be done in an Administration Server instance of WebLogic Server because that is what will be used in a production environment.

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 Section 52.5, "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 (ApplCore) domain extension templates so all of the Oracle Fusion applications will run on Integrated WebLogic Server as they would in an Administration Server instance of WebLogic Server.

JDeveloper has a default connection to Integrated WebLogic Server and does not require any deployment profiles or descriptors.

When you run your application in JDeveloper using the **run** or **debug** commands, Integrated WebLogic Server starts automatically and your application runs in the target browser.

When you use JDeveloper to run an application for the first time, it automatically creates the Integrated WebLogic Server instance.

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 52–5.

Figure 52–5 Terminating the Application

To terminate the server, select the server name, as shown in Figure 52–6.

Figure 52–6 Terminating the Integrated WebLogic Server Instance

52.2.1 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.

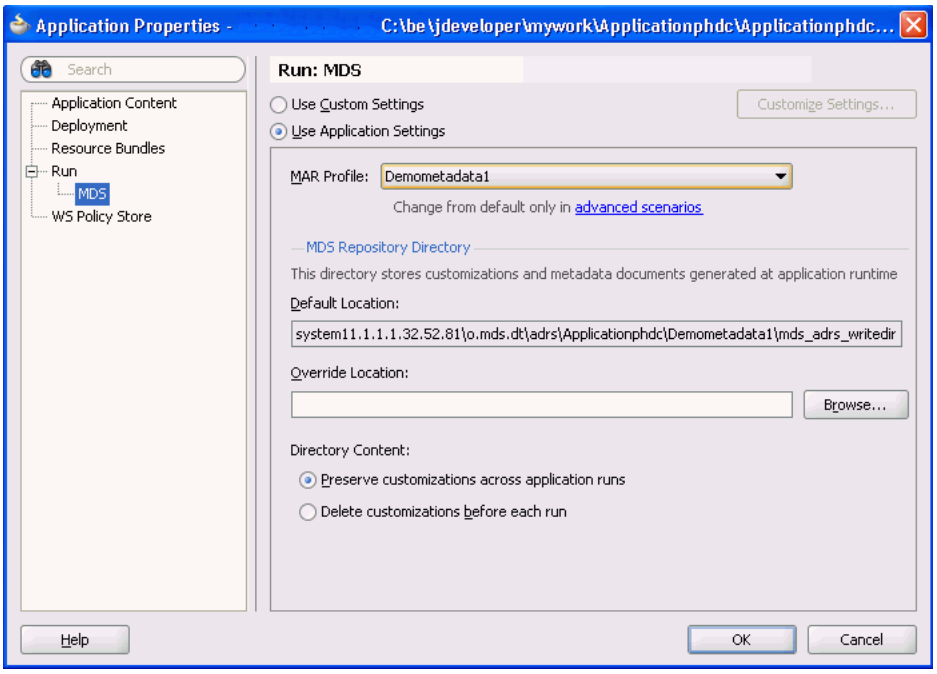
Any customizations or documents created by the application are written to this simulated MDS repository directory. You can keep the default location for this directory or you can set it to a different directory. You also have the option to preserve customizations across different application runs or 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 Section 52.3.2, "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 52–7.

Figure 52–7 Setting the Run MDS Options



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.

52.3 Preparing to Deploy Oracle ADF Applications to an Administration Server Instance of WebLogic Server

You must prepare the application and the WebLogic Server instance before you deploy applications to an Administration Server instance of WebLogic Server.

Before you begin:

Before you deploy the application to a standalone WebLogic Server instance, you need to:

- Create and configure the WebLogic Server domains using the Configure Fusion Domain wizard as described in Chapter 2, "Setting Up Your Development Environment."
- If the application is using ADF Security, you may need to:
  - Configure for Oracle Single Sign-on using Oracle Access Manager (OAM). For more information, see the "Applications That Will Run Using Oracle Single Sign-On (SSO)" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
  - Migrate application-level security information to the WebLogic Server instance. For more information, see the "Configuring Security for WebLogic

Server" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

- Set up JDBC URL for WebLogic Server. For more information, see the "Applications with JDBC URL for WebLogic" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
- Set up JDBC datasource for WebLogic Server. For more information, see the "Applications with JDBC Data Source for WebLogic" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
- Understand LDAP-based stores. For more information, see Section 49.3.8, "What You May Need to Know About Testing."

52.3.1 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.

52.3.2 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. Once 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
- EJB JAR deployment profile. For instructions, see the "Creating an EJB JAR Deployment Profile" section of the *Oracle Fusion Middleware Java EE Developer's Guide for Oracle Application Development Framework*.
- WAR deployment profile. For instructions, see the "Creating a WAR Deployment Profile" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- MAR deployment profile. For instructions, see the "Creating a MAR Deployment Profile" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- EAR deployment profile. For instructions, see the "Creating an Application-Level EAR Deployment Profile" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*

52.4 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 or Ant commands.

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 Administrator's 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 to configure MDS before deploying the EAR file. 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 WebLogic Scripting Tool Command Reference*.

52.4.1 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:

1. Start WebLogic Server instance.
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.

52.4.2 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***.

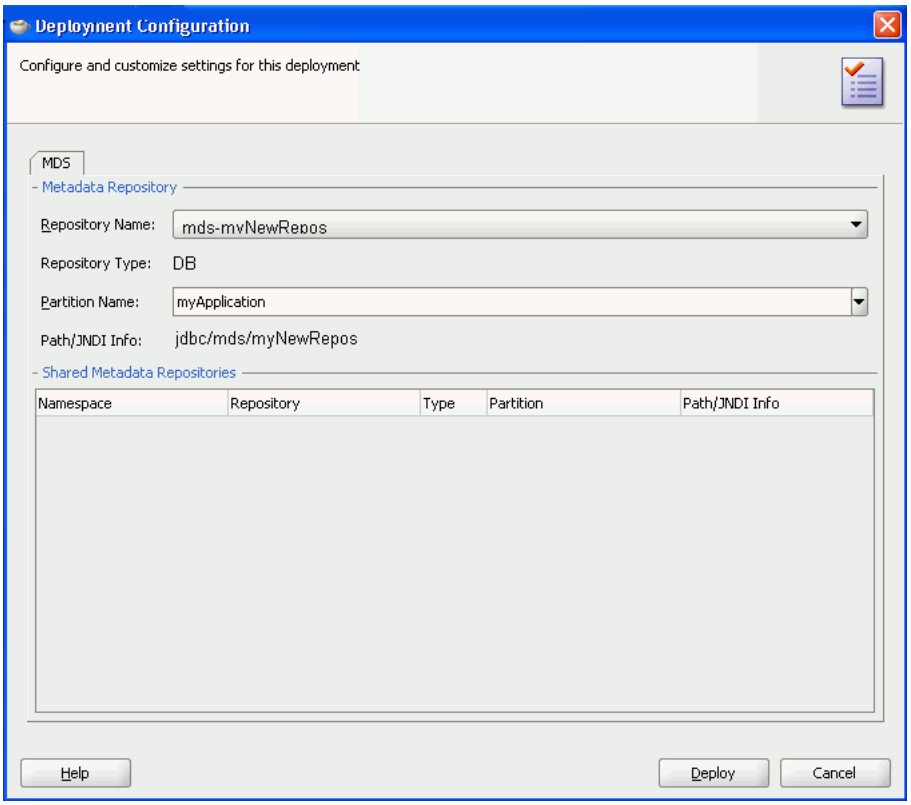
   b. In the Deploy wizard Deployment Action page, select **Deploy to Application Server** and click **Next**.

   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 52–8.

Figure 52–8 MDS Configuration and Customization For Deployment



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 Administrator's Guide*.

Note: If you are deploying an Oracle ADF application, do not use the **Deploy to all instances in the domain** option.

---

e. Click **Finish**.

2. Verify the run-time application as:

   http://httpHost:httpPORT/<CONTEXT>/faces/<landing jspx

   For example,
   http://server06.us.company.com:7001/UIPatternsDemo/faces/ServiceRequest

   http://server12.company.com:7001/D7Build1-ViewController-context-root/faces/TreeHomePage.jspx

52.4.3 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**.

52.5 Deploying 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 52–1.

52.5.1 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 the "To create an application server connection" section of the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.
2. Deploy the Project. For instructions to deploy a SOA project, see the "Deploying the Profile " section of the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*

3. Check the deployed SOA Project.

52.5.1.1 Check the Deployed SOA Project

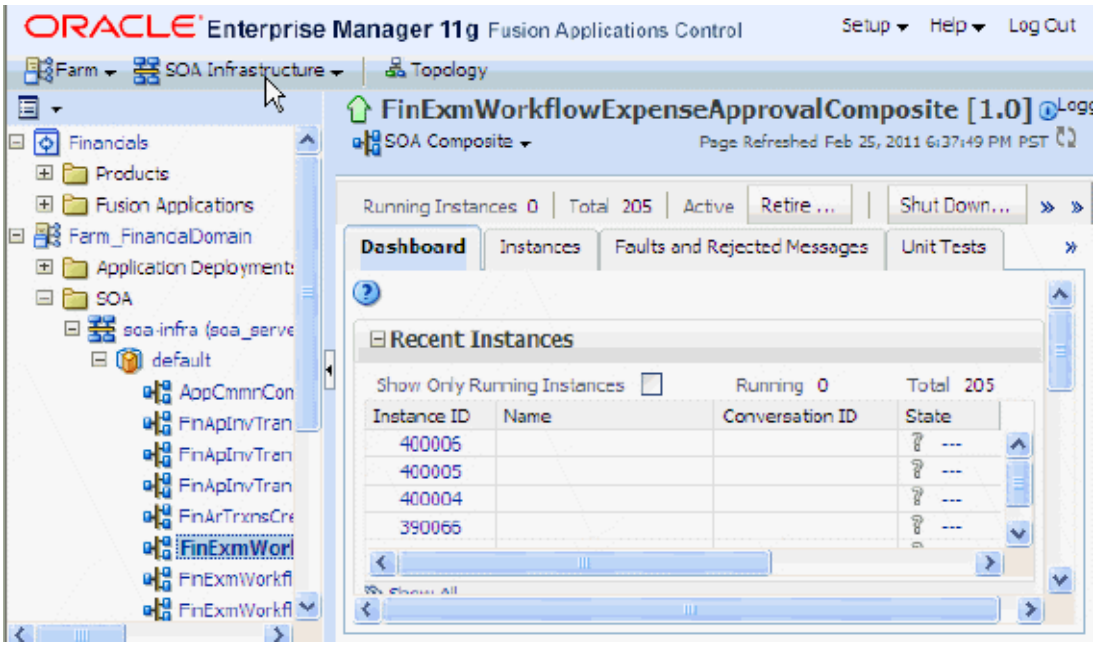
You can check and run your deployed SOA project from two locations:

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.us.oracle.com:7001/em
```

After you log in, a screen similar to Figure 52–9 displays (the SOA tree in the left pane has been expanded and the first deployment has been selected):

Figure 52–9 Checking SOA Deployment from the EM



53 Creating Repository Connections

This chapter provides information about the Oracle Content Server, Oracle Data Integrator (ODI), and Oracle Business Activity Monitoring 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:

53.1 Creating a Content Repository Connection

Oracle Content Server, which serves as the base for the Oracle Universal Content Management (Oracle UCM) 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 JDeveloper, you must set up a content server-based content repository connection.

For more information about Attachment components, see Chapter 18, "Implementing Attachments."

For more information about Oracle Content Server, see *Oracle Fusion Middleware User's Guide for Oracle Content Server*.

For more information about content integration, see the "Introduction to Integrating and Publishing Content" chapter of the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

53.1.1 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.

53.1.1.1 Creating a Connection for Oracle Fusion Applications Development

To set up a content server-based repository connection from an Oracle Fusion application, you run WebLogic Server Tool (WLST) commands to synchronize the Oracle WebLogic server credential store with the Oracle Content Server credential

store, and then you use the Create Content Repository Connection wizard to set up a content server-based content repository connection.

For information about using the WLST command-line scripting interface, see *Oracle Fusion Middleware Oracle WebLogic Scripting Tool*.

Before you begin:

- Verify that Oracle 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 Section 18.2, "Creating Attachments." When the application is deployed, the policies in `jazn-data.xml` are merged into the `system-jazn-data.xml` file in `$DOMAIN_HOME/config/fmwconfig` on Oracle WebLogic Server.
- Log into 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 WebLogic Server domain's default key store located at `<WLS DOMAIN HOME>/config/fmwconfig/default-keystore.jks`.
2. Replace the `<WLS DOMAIN HOME>/config/fmwconfig/default-keystore.jks` file with a copy of the Oracle UCM server's domain default key store, which is located at `<Oracle UCM 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_HOME/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 the 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 the WLST tool, execute the following commands to store the credentials. The user names and passwords must be the same as for the Oracle UCM domain. For more information about the Oracle UCM key store credentials, see the "Configuring Oracle Enterprise Content Management Suite" chapter in the *Oracle Fusion Middleware Installation Guide for Oracle Enterprise Content Management Suite*.

   When executing the commands, replace *user name* and *password for user* with the user names and passwords that are used in the Oracle UCM 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:

```
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 53–1, complete the following information:

   Create Connection In: Select **Application Resources**.

   Connection Name: Enter `FusionAppsContentRepository`.

   Repository Type: Select **Oracle Content Server**.

   Set as Primary Connection for Document Library: Select this check box.

   Configuration Parameters: Enter values for the parameters listed in Table 53–1. If a parameter is not listed in the table, leave the value blank.

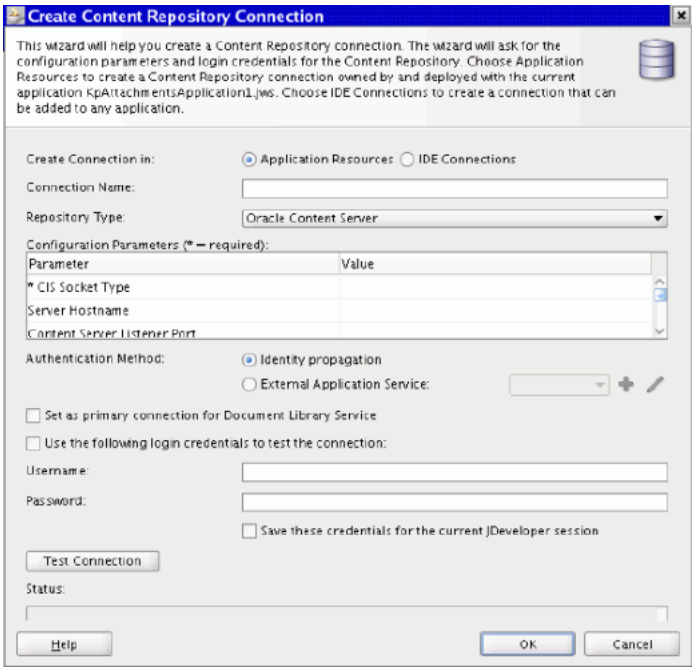
   Contact your system administrator to obtain the correct information.

Table 53–1 Example Configuration Parameters

Parameter	Value
RIDC Socket Type	`jaxws`
Admin Username	The name of a user who has been granted administration privileges on the Oracle UCM server.
Web Server Plugin	The idcnativews web service that is defined on the Oracle UCM server. This is typically `http://`*host*`:`*port*`/idcnativews`. Check with your system administrator.

   Authentication Method: Select **Identity propagation**.

Figure 53–1 Create Content Repository Connection



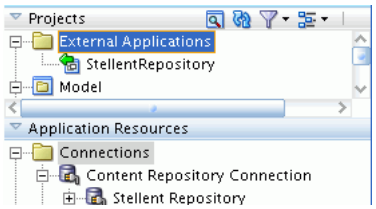
8. Click **Test Connection** and verify that the status returned is `Success!`

> **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 the **Connections** node to see your new content repository connection, as shown in Figure 53–2.

Figure 53–2 JDeveloper — Application Navigator



53.1.1.2 Creating 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 Oracle UCM environment. In this situation, you 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 Section 18.2, "Creating Attachments." When the application is deployed, the policies in `jazn-data.xml`

are merged into the `system-jazn-data.xml` file in `$DOMAIN_HOME/config/fmwconfig` on Oracle WebLogic Server.

- Ensure that the sockets on Oracle UCM Managed Server are enabled. For more information, see the "Completing the Oracle UCM Configuration" section in the *Oracle Fusion Middleware Installation Guide for Oracle Enterprise Content Management Suite*.
- You need the following information to create a content repository connection:
  - Host name of the machine that is running the Oracle UCM Managed Server.
  - Oracle UCM Managed Server listener port.
  - URL for the login page to the content server instance.
  - Oracle UCM shared credentials user name and password

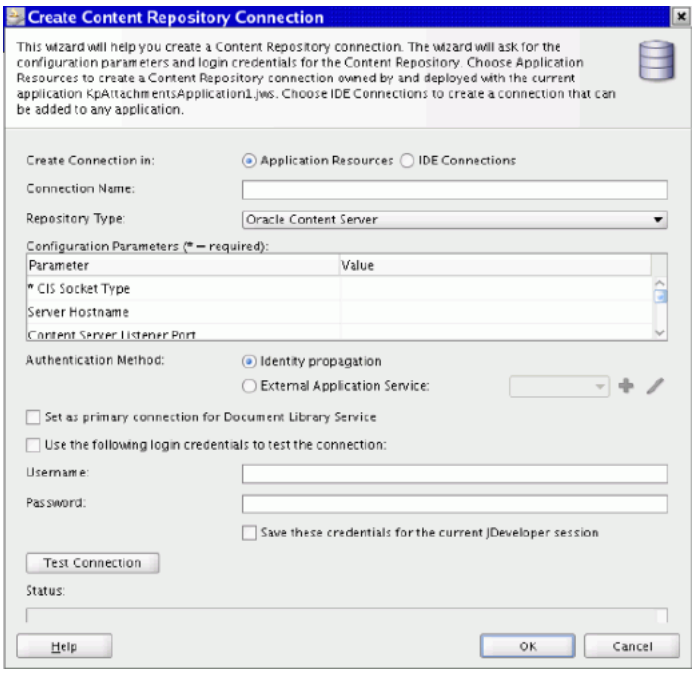
  Contact your system administrator to obtain this information.

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 53–3.

Figure 53–3 Create Content Repository Connection



2. Complete the following information:

   Create Connection In: Select **Application Resources**.

   Connection Name: Enter the appropriate name for this connection.

   Repository Type: Select **Oracle Content Server**.

   Set as Primary Connection for Document Library: Select this check box.

Configuration Parameters: Enter values for the parameters shown in Table 53–2. With the exception of 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 53–2 Example Configuration Parameters

Parameter	Value
RIDC Socket Type	socket
Server Hostname	abc.example.com
Content Server Listener Port	4444

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 53–4.

Login URL: Paste the URL for the Oracle UCM Server login page. (Contact your system administrator for this information). For example:

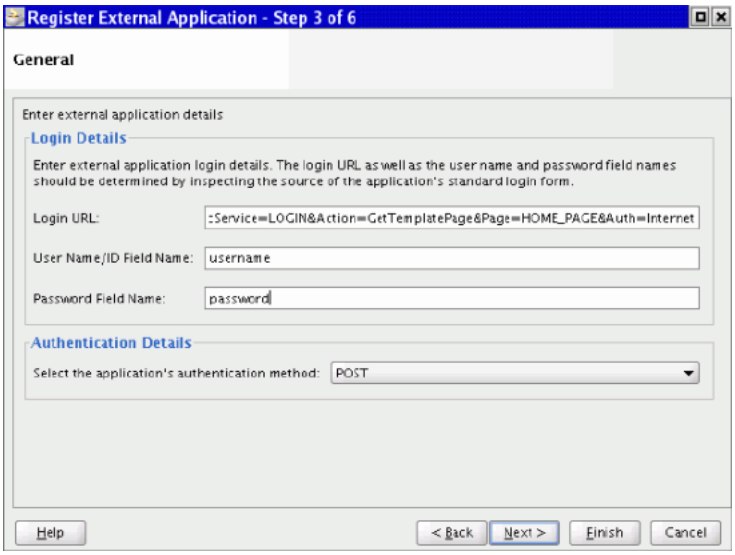
```
http://abc.example.com:2244/abc/abcplg?AbcService=LOGIN&Action=GetTemplatePage&Page=HOME_PAGE&Auth=Internet
```

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 53–4 Register External Application Wizard — General (Step 2)



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.

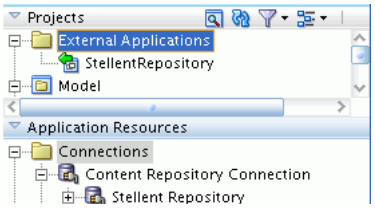
3. Click **Test Connection** and verify that the status returned is *Success!*

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 the **Connections** node to see your new content repository connection, as shown in Figure 53–5.

Figure 53–5 JDeveloper — Application Navigator



53.1.2 Troubleshooting Content Server Connections

Exceptions can occur when the connection is improperly configured. The three most common exceptions are the following errors:

- Insufficient user privileges error
- WS-Security header processing error
- Access denied error

For more information, see the "Diagnosing Problems" chapter in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

53.1.2.1 User Does Not have Sufficient Privileges

If you are sure that the user is a member of the AttachmmentsUser 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 "Key store 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 Section 18.2, "Creating Attachments" have been added to the `jazn-data.xml` file. For information about applications logging, see the "Introduction to Troubleshooting Using Incidents, Logs, QuickTrace, and Diagnostic Tests" chapter in the *Oracle Fusion Applications 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`, as described in the "System Audit Tracing Sections Information" section in the *Oracle Fusion Middleware System Administrator's Guide for Oracle Content Server*. Be sure to enable full verbose tracking and enable save. Access the system audit information server output and search for `XFND_SIGNATURE`, as described in the "System Audit Information" section in the *Oracle Fusion Middleware System Administrator's Guide for Oracle Content Server*. 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.

53.1.2.2 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.

53.1.2.3 Access Denied: Credential AccessPermission

The following exception typically indicates that the code grants described in Section 18.2, "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_domain*`/config/fmwconfig` on Oracle WebLogic Server.

```
access denied (oracle.security.jps.service.credstore.CredentialAccessPermission
context=SYSTEM,mapName=oracle.wsm.security,keyName=enc-csf-key read)
```

53.2 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 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.

To learn how to connect to the ODI master and work repositories, see the "Administering the Oracle Data Integrator Repositories" chapter in the *Oracle Fusion Middleware Developer's Guide for Oracle Data Integrator*.

For information about the role of ODI in securing Oracle Fusion applications, see Section 46.1.1.7, "Oracle Data Integrator,"

53.3 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.

For more information about Oracle Business Activity Monitoring, see Oracle Fusion Middleware User's Guide for Oracle Business Activity Monitoring.

You must create a connection to an 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 the Oracle BAM data objects. Only one Oracle BAM Server connection per BPEL project is currently supported.

When building an application in JDeveloper, the methods of connecting to the Oracle BAM server are:

- Oracle BAM Adapter in a service-oriented architecture (SOA) composite application
- Oracle BAM sensor actions in a BPEL process

53.3.1 How to Create an Oracle BAM Connection

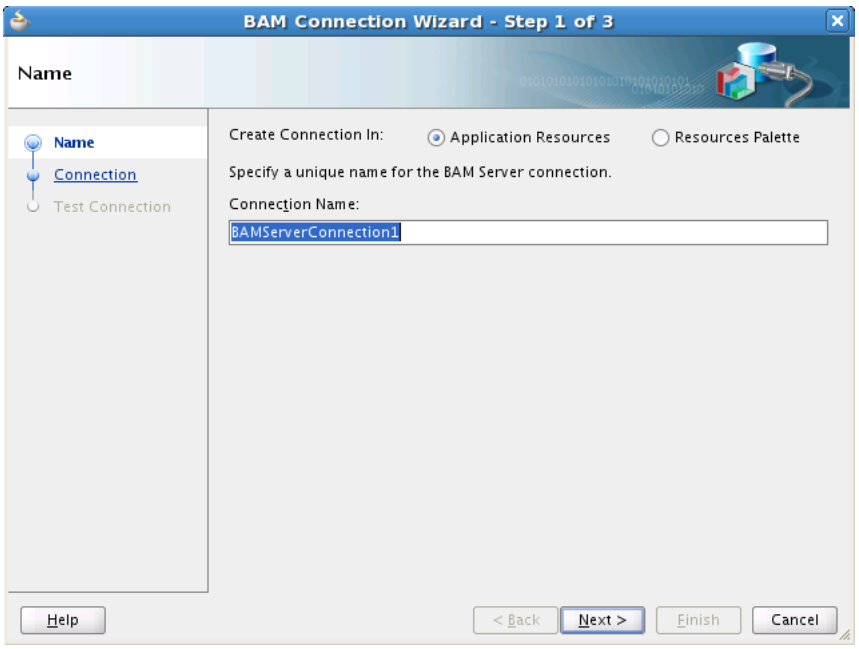
You use the BAM Connection wizard to create an Oracle BAM connection.

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 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 53–6.

Figure 53–6 BAM Connection Wizard — Name Page



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 the Oracle BAM Report Server and web server are installed. In most cases, the Oracle BAM web host and Oracle BAM server host are the same.

   BAM Server Host: Enter the name of the host on which the Oracle BAM Server is installed.

   User Name / 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 the Oracle BAM report cache, which is part of the 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**.

53.3.2 How to Use Oracle BAM Adapter in a SOA Composite Application

The Oracle BAM Adapter is a (Java EE Connector Architecture) JCA-compliant adapter, which can be used from a J2EE client to send data and events to the Oracle BAM Server. The Oracle BAM Adapter supports the following operations on Oracle BAM data objects: inserts, updates, upserts, and deletes. The Oracle BAM Adapter can perform these operations over EJB calls or over Simple Object Access Protocol (SOAP), all configurable in Oracle 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.

The Oracle BAM adapter provides three mechanisms by which you can send data to an Oracle BAM server in your SOA composite application as you develop it in Oracle JDeveloper.

- The 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 the Oracle BAM Adapter.
- The Oracle BAM Adapter can also be used as a partner link in a BPEL process to send data to a 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 the Oracle BAM data objects.

For more information about the Oracle BAM Adapter in a SOA composite application, see the "Integrating Oracle BAM With SOA Composite Applications" chapter in the Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.

53.3.3 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 an Oracle BAM Server. When you create the sensor action, you can select an Oracle Application 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 *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

54

Defining Profiles

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:

54.1 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 Section 54.4, "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.

54.2 Integrating Profiles 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 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 Chapter 49, "Implementing Function Security."

For more information about task flows, see the*Oracle Fusion Applications Common Implementation Guide*.

Table 54–1 lists the task flows related to profiles and their parameters.

Table 54–1 Profiles Task Flows and Parameters

Task Flow Name	Task Flow XML	Parameters Passed	Behavior	Comments
Manage Administrator Profile Values	/WEB-INF/oracle/apps/fnd/applcore/profiles/ui/flow/ManageAdminProfileValuesTF.xml#profileValues_task-flow-definition	mode='search' [moduleType] [moduleKey] [categoryName] [categoryApplicationId] mode='edit' profileOptionName [pageTitle]	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.	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.
Manage Profile Categories	/WEB-INF/oracle/apps/fnd/applcore/profiles/ui/flow/ManageProfileCategoriesTF.xml#profileCategories_task-flow-definition	mode='search' [moduleType] [moduleKey] mode='edit' name applicationId [pageTitle]	Search and edit profile categories. To search all profile categories, do not pass any parameters. To search all profile categories in a module, pass in moduleType/moduleKey. To edit a specific profile category, pass in name/applicationId.	Search and edit profile categories. To search all profile categories, do not pass any parameters. To search all profile categories in a module, pass in moduleType/moduleKey. To edit a specific profile category, pass in name/applicationId.
Manage Profile Options	/WEB-INF/oracle/apps/fnd/applcore/profiles/ui/flow/ManageProfilesTF.xml#profiles_task-flow-definition	mode='search' [moduleType] [moduleKey] mode='edit' profileOptionName [pageTitle]	Search and edit profile definitions. In 'search' mode: To search all profile options, do not pass any parameters. To search all profile options in a module, pass in moduleType/moduleKey. In 'edit' mode: To edit a specific profile option, pass in profileOptionName. If mode is not explicitly passed, the default is 'search'.	Search and edit profile definitions. In 'search' mode: To search all profile options, do not pass any parameters. To search all profile options in a module, pass in moduleType/moduleKey. In 'edit' mode: To edit a specific profile option, pass in profileOptionName. If mode is not explicitly passed, the default is 'search'.

54.3 Setting and Accessing Profile Values

You can set profile values using the Setup UI, and access them programmatically or by using expression language.

54.3.1 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 54–1.

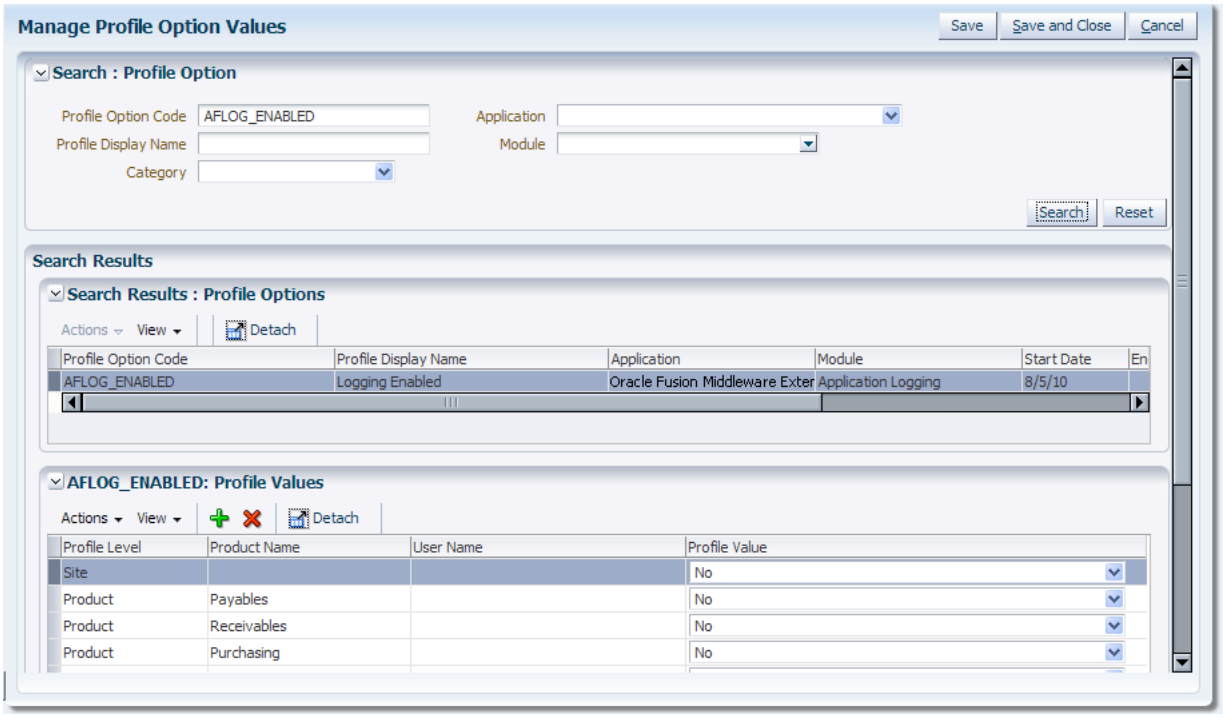
Notes:

- 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:

1. Go to the Manage Profile Option Values page to search for the required profile option.

Figure 54–1 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.

54.3.2 How to Access 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 Section 3.3, "Adding the Applications Core Library to Your Data Model Project."

To access profile values programmatically:

Import the Profile class and call the `Profile.get()` method to get the values for the profile name provided.

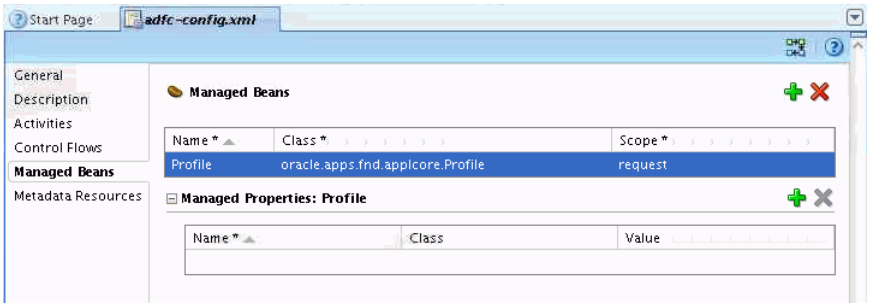
For example:

```
import oracle.apps.fnd.applcore.Profile;

...
fndDiagnostics = Profile.get("AFLOG_ENABLED");
```

54.3.3 How to Access 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 54–2.

Figure 54–2 Edit adfc-config.xml File



Once the bean is defined, you can refer to any profile value as:

```
# {Profile.values.PROFILE_OPTION_NAME}
```

54.4 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. At this time, 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 54–2 describes how profile settings are used:

Table 54–2 Profile Settings

Hierarchy	Level	Profile Setting
1 (Lowest)	Site	All users at an installation site.
2	Product	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.
3 (Highest)	User	An individual user, identified by their UserID (UserGUID) for the current session.

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. As a result, in a multi-tenant environment, a site-level profile value behaves like an enterprise-level profile value.

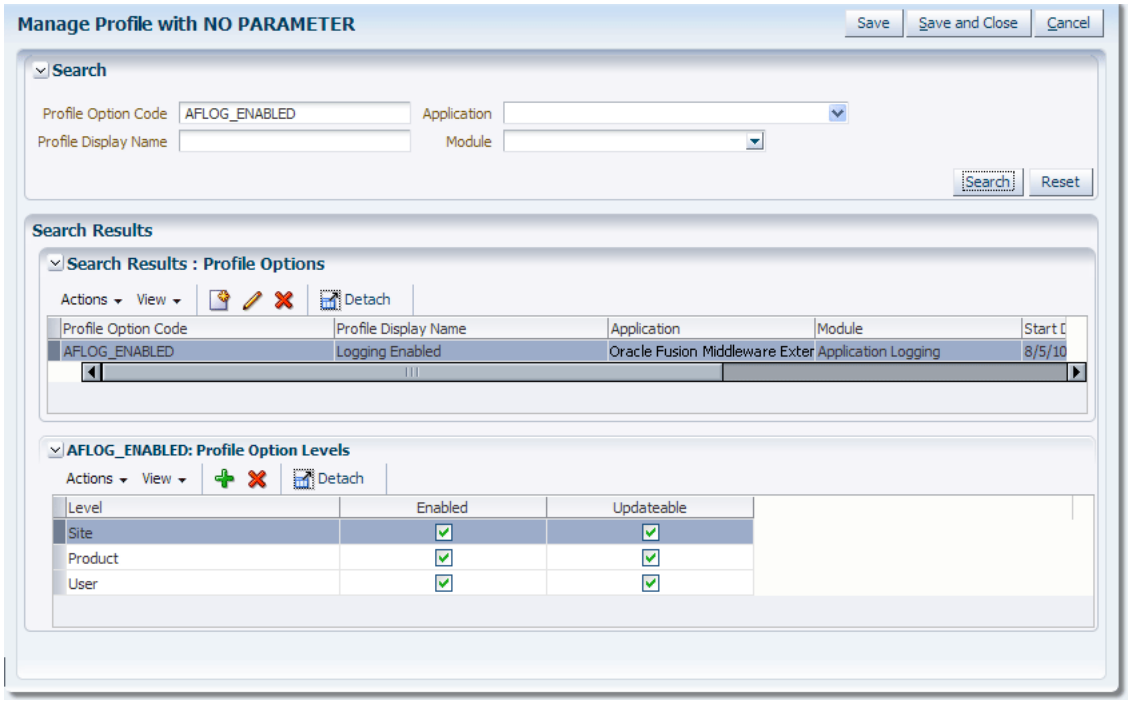
54.4.1 How to Edit Profile Definitions

You can use the Profile editor to update profile definitions. Figure 54–3 shows the Manage Profiles Options page.

To edit profile definitions:

1. Go to the Manage Profiles Options page to search for the profile that you want to update.

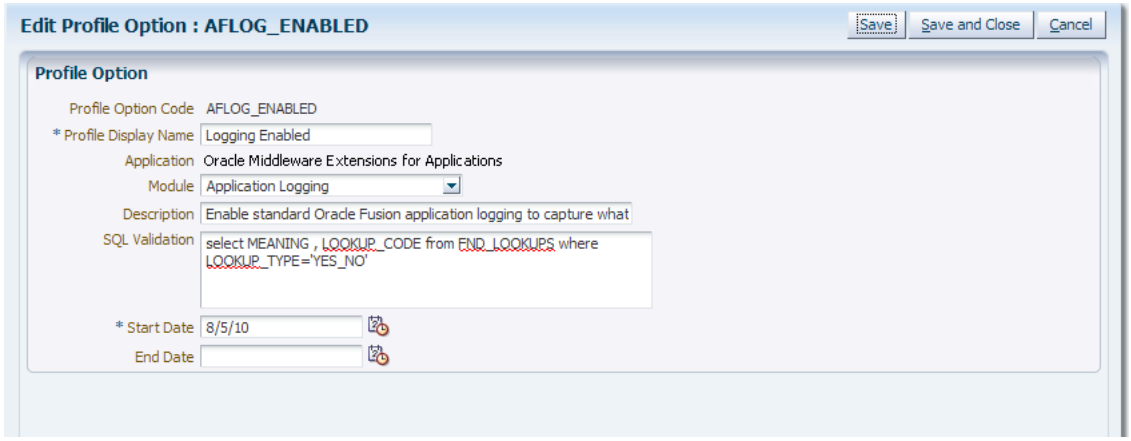
Figure 54–3 Manage Profiles Options Page



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 54–4.

Figure 54–4 Edit Profile Option



   - **Delete** an existing profile option
   - **Detach** a profile option to open it in a new window

3. In the **Search Results: Profile Option Levels** section, do any of the following:
   - Use the **Actions** or **View** options
   - **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 or not 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

54.4.2 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'
```

54.5 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 categories out of the box; customers are free to create their own or edit those that are shipped.

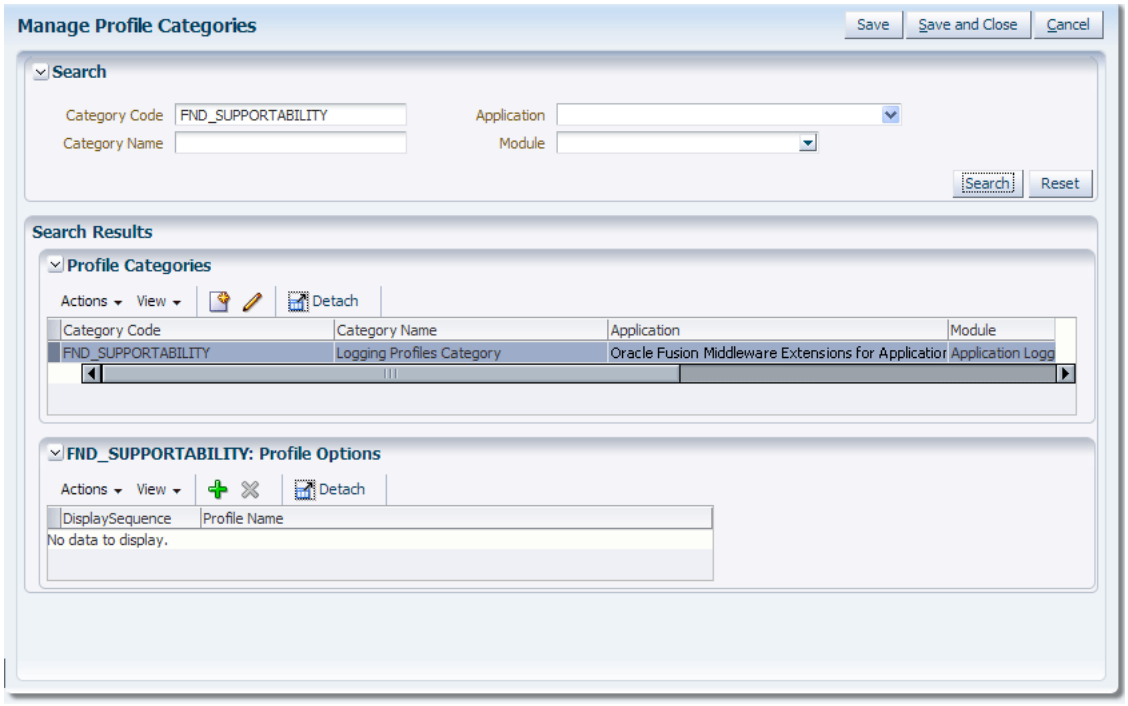
54.5.1 How to Manage Profile Categories

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:

1. Go to the Manage Profile Categories page and search for the required profile category, as shown in Figure 54–5.

Figure 54–5 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 54–6.

Figure 54–6 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

55

Initializing Oracle Fusion Application Data Using the Seed Data Loader

This chapter discusses the Seed Data Loader and using it from within Oracle JDeveloper.

This chapter includes the following sections:

- Section 55.1, "Introduction to the Seed Data Loader"
- Section 55.2, "Using the Seed Data Loader in JDeveloper"
- Section 55.3, "Translating Seed Data"

55.1 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 needs to 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 55–1, " Available Seed Data Loaders".

55.2 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.

55.2.1 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 Section 55.2.2, "How to Set Up the Seed Data Environment".

Available Seed Data Loaders

The loaders and view objects listed in Table 55–1 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 55–1 Available Seed Data Loaders

Loader	View Object	Is the Module Striped?
`attachments.attachmentLoader.applicationModule.attachmentsLoaderAM`	FndDocumentEntitiesVL	N
`attachments.attachmentLoader.applicationModule.attachmentsLoaderAM`	FndDocumentCategoriesVL	Y
`attachments.attachmentLoader.applicationModule.attachmentsLoaderAM` **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.	FndDocCategoriesToEntitiesVO	N
`dataSecurity.dataSecurityService.applicationModule.DataSecurityAM`	FndMenus	Y
`dataSecurity.dataSecurityService.applicationModule.DataSecurityAM`	FndGrants	Y
`dataSecurity.dataSecurityService.applicationModule.DataSecurityAM`	FndObjects	Y
`dataSecurity.dataSecurityService.applicationModule.DataSecurityAM`	FndFormFunctions	Y

Table 55–1 (Cont.) Available Seed Data Loaders

Loader	View Object	Is the Module Striped?
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
lookups.lookupService.applicationModule.LookupServiceAM	CommonLookupType1	Y
lookups.lookupService.applicationModule.LookupServiceAM	SetIdLookupType1	Y
messages.messageService.applicationModule.MessageServiceAM	Message	Y
nls.currencyService.applicationModule.CurrencyServiceAM	Currency	N
nls.isoLanguageService.applicationModule.IsoLanguageServiceAM	IsoLanguage	N
nls.languageService.applicationModule.LanguageServiceAM	Language	N

Table 55–1 (Cont.) Available Seed Data Loaders

Loader	View Object	Is the Module Striped?
`nls.naturalLanguageService.applicationModule.NaturalLanguageServiceAM`	NaturalLanguage	N
`nls.territoryService.applicationModule.TerritoryServiceAM`	Territory	N
`nls.timezoneService.applicationModule.TimezoneServiceAM`	Timezone	N
`profiles.profileService.applicationModule.ProfileServiceAM`	ProfileCategory	Y
`profiles.profileService.applicationModule.ProfileServiceAM`	ProfileHierarchy	N
`profiles.profileService.applicationModule.ProfileServiceAM`	ProfileLevel	N
`profiles.profileService.applicationModule.ProfileServiceAM`	ProfileOption	Y
`ref.industryService.applicationModule.IndustryServiceAM`	Industry	N
`setid.setIdService.applicationModule.SetIdServiceAM`	SetIdSet	N
`setid.setIdService.applicationModule.SetIdServiceAM`	SetIdSummary	Y
`setid.setIdService.applicationModule.SetIdServiceAM`	SetIdAssignment	N
`setid.setIdService.applicationModule.SetIdServiceAM`	SetIdReferenceGroup	Y
`trees.loader.applicationModule.TreeStructureLoader`	FndTreeStructure	Y
`taxonomy.taxonomyService.applicationModule.ApplTaxonomyAM`	ApplTaxonomyVO	Special case - all the data extracted into a single file.
`taxonomy.taxonomyService.applicationModule.ApplTaxonomyAM`	ApplTaxonomyHierarchyVO	Special case - all the data extracted into a single file
`taxonomy.taxonomyService.applicationModule.ApplTaxonomyAM`	ApplTaxonomyPVO	
`taxonomy.taxonomyService.applicationModule.ApplTaxonomyAM`	ApplTaxonomySeedDataVO	
`taxonomy.taxonomyService.applicationModule.ApplTaxonomyAM`	ApplTaxonomySeedDataPVO	
`taxonomy.taxonomyService.applicationModule.ApplTaxonomyAM`	ApplTaxonomyComponentsVO	Y

Table 55–1 (Cont.) Available Seed Data Loaders

Loader	View Object	Is the Module Striped?
`taxonomy.taxonomyService.applicationModule.ApplTaxonomyAM`	ApplTaxonomyNodeComponentsVO	Y
`trees.loader.applicationModule.DefaultTreeLoader`	FndTree	N
`oracle.apps.fnd.appltest.diagfwk.seeddata.model.DiagFwkSeedDataAM`	FndDiagTag	Y
`oracle.apps.fnd.appltest.diagfwk.seeddata.model.DiagFwkSeedDataAM`	FndDiagTest1	Y

55.2.2 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.

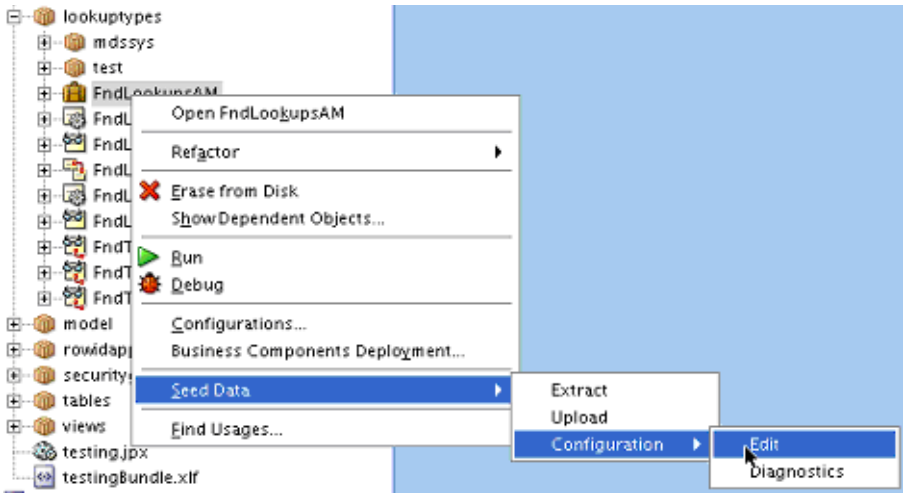
2. Create the ADF Business Components artifacts to represent the logical data model, including entity objects and view objects in the project. Create the relationships between the entity objects and view objects in accordance with the Applications Standards. Add the objects to an application module that 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 55–1.

Figure 55–1 Starting Seed Data Configuration

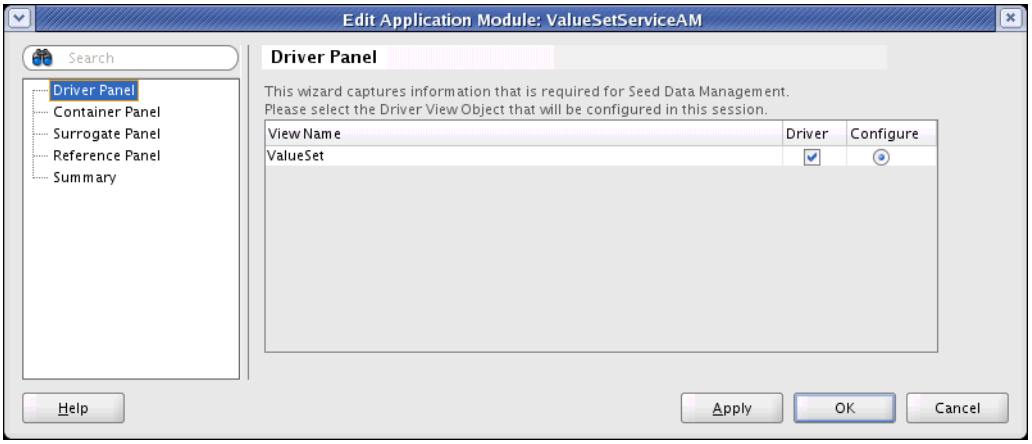


Note: If the seed application module in use is derived from a subclass of `OAApplicationModuleImpl`, that sub-class and its dependencies, if any, should be made available in compiled form using the Libraries/Classpath feature of JDeveloper. If this step is not done, extract/upload will fail.

The Seed Data Configuration wizard launches and displays the Driver Definition Panel.

This panel, shown in Figure 55–2, initially shows the available view objects in the root level of the application module data model that can serve as Driver view objects.

Figure 55–2 Seed Data Configuration Driver Panel



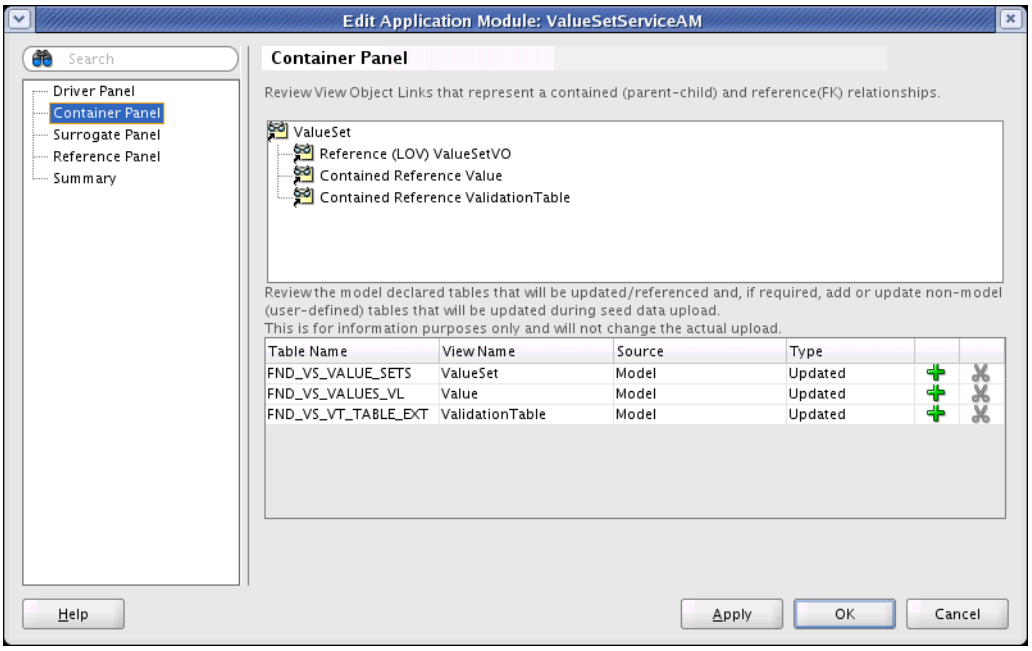
a. Select the **Driver** check box for those view objects to serve as the Driver for Seed Data Extract. More than one Driver can exist within an application module; however, only one Driver can be specified at Extract time.

b. Select the **Configure** radio button for the Driver view object to configure 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 55–3, 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 55–3 Seed Data Configuration Container Panel



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/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 is required to 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 could also 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: 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.

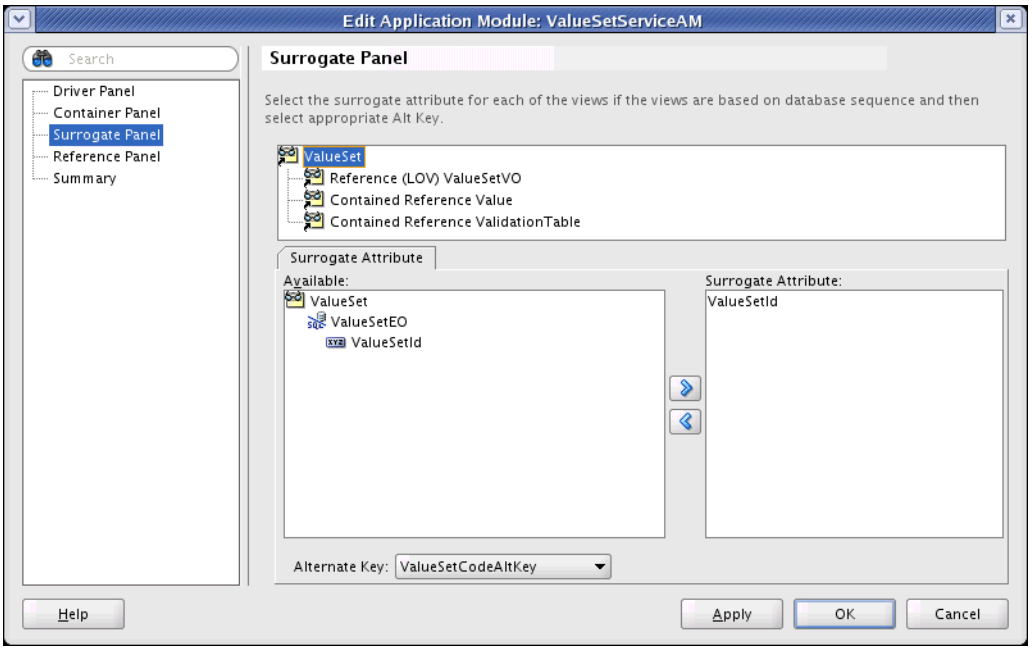
Source: Shows the source of this definition. The value of **M** is reserved to indicate that the Table Name is derived from the ADF Business Components Model. You can define your own definitions for additional table entries.

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 55–4, to declare surrogate attributes of the view objects of the Seed Data Configuration.

Figure 55–4 Seed Data Configuration Surrogate Panel



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/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 *Oracle Fusion Middleware Fusion Developer's Guide for 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 PK has many attributes and one of them is being marked as a surrogate, all the other attributes in the PK, 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 PK to appear in the list.

To declare an attribute as a Surrogate Attribute:

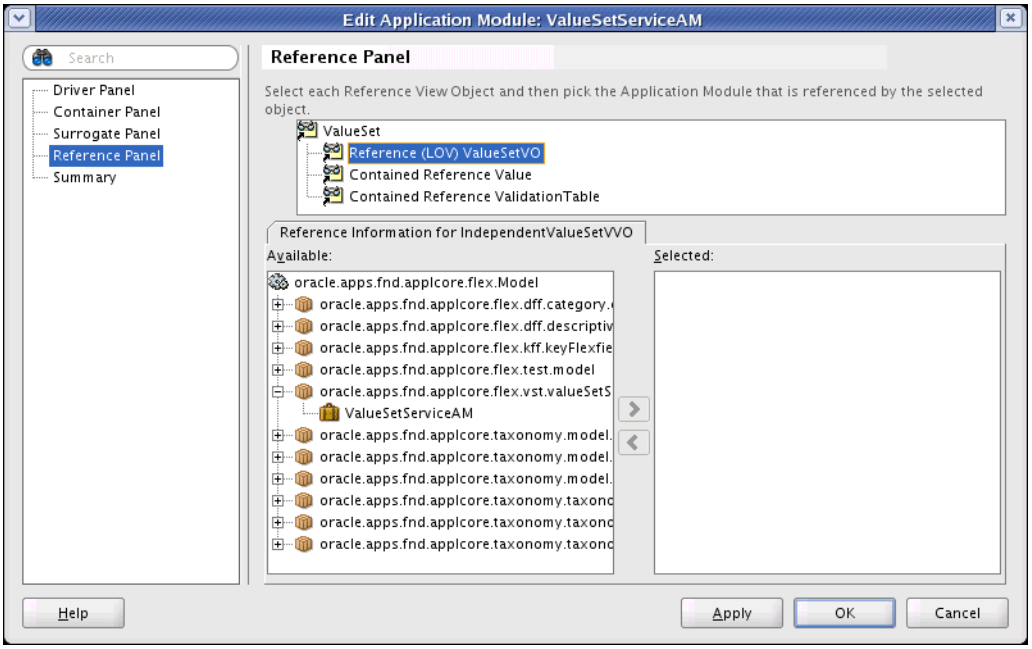
a. Select the view object that contains the surrogate attribute. The left bottom panel will show the list of attributes that are potential surrogate attributes.

b. Select the Surrogate Attribute and click the shuttle button.

Click **Reference Panel**.

Use the Reference Panel, shown in Figure 55–5, to declare the Data Upload Mode for the Seed Data Configuration. Provide reference information on reference view objects.

Figure 55–5 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 55–2, " Service Foreign Key LOV Scenarios" 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 55–2 Service Foreign Key LOV Scenarios

Scenario	What needs to be done
#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.)	An LOV should be defined on the alternate key attribute, with the foreign key ID as a derived attribute. **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: `<ViewObject xmlns="http://xmlns.oracle.com/bc4j" Name="SdLovrefReferringVO" Version="11.1.1.53.68" SelectList="SdLovrefReferringEO.ID, SdLovrefReferringEO.REFERRING_DATA, SdLovrefReferredEO.REFERRED_ALTATTR_2, SdLovrefReferredEO.REFERRED_ALTATTR_1, SdLovrefReferredEO.REFERRED_ID, SdLovrefReferringEO.REFERENCE_ID" FromList="SD_LOVREF_REFERRING SdLovrefReferringEO, SD_ LOVREF_REFERRED SdLovrefReferredEO" BindingStyle="OracleName" CustomQuery="false" RowClass="oracle.apps.fnd.applcore.oaext.model.OAViewRowIm pl" ComponentClass="oracle.apps.fnd.applcore.oaext.model.OAVie wObjectImpl" PageIterMode="Full" LBThrowOnMisMatch="true" UseGlueCode="false" Where="SdLovrefReferringEO.REFERENCE_ID = SdLovrefReferredEO.REFERRED_ID(+)">`

Table 55–2 (Cont.) Service Foreign Key LOV Scenarios

Scenario	What needs to be done

#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 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.
	2. Define the LOV view object (OrganizationVVO).
	3. Define a view accessor on EmpEO/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 the PersonVO.xml to make the LOVs driven by all the foreign alternate key attributes, such as:
	`<ListBinding`
	`Name="LOV_OrganizationName"`
	`ListVOName="OrganizationVA"`
	`ListRangeSize="-1"`
	`NullValueFlag="none"`
	`NullValueId="LOV_OrganizationName_LOVUIHints_NullValueId"`
	`MRUCount="0">`
	`<AttrArray Name="AttrNames">`
	`<Item Value="OrganizationName"/>`
	`<Item Value="BusinessGroupName"/>`
	`</AttrArray>`
	`<AttrArray Name="DerivedAttrNames">`
	`<Item Value="OrganizationId"/>`
	`</AttrArray>`
	`<AttrArray Name="ListAttrNames">`
	`<Item Value="OrganizationName"/>`
	`<Item Value="BusinessGroupName"/>`
	`<Item Value="OrganizationId"/>`
	`</AttrArray>`
	`...`
	6. Manually edit the VO.xml and add LBThrowOnMisMatch="true"

Table 55–2 (Cont.) Service Foreign Key LOV Scenarios

Scenario	What needs to be done
#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) needs to be resolved first, either based on Scenario #1 or #2. Then the second alternate key should filter by the first alternate key. 1. Define an LOV view object based on CountryEO. 2. Define an LOV view object based on CityEO. Define a view criteria to filter by CountryId. 3. Define a view accessor on EmpEO/PersonVO pointing to CountryVVO. 4. Define a view accessor on EmpEO/PersonVO pointing to CityVVO, and bind CountryId to BirthOfCountry. 5. Define an LOV on CountryName using CountryVVO view accessor, with BirthOfCountry as a derived attribute from CountryId from CountryVVO. 6. Define an LOV on CityName using CityVVO view accessor, with BirthOfCity as a derived attribute from CityId from CityVVO. 7. Manually edit the VO.xml and add LBThrowOnMisMatch="true"
#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 needs to 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.

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 attribute(s). To do this, you need a "fake" derived attribute defined on the LOV.

For example:

```
LinesStatusEO(LineStatusId, StatusCode)
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.

55.2.3 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 also. 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:

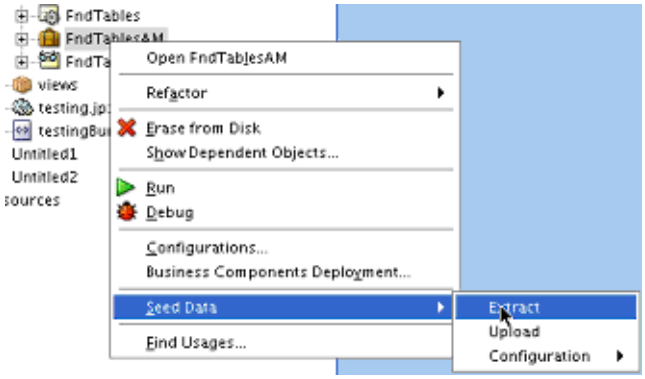
- From JDeveloper using the Seed Data Framework Console
- By using an external Command Line Interface, as shown in Section 55.2.4.2, "Using the Extract Seed Data Command Line Interface"

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 55–6.

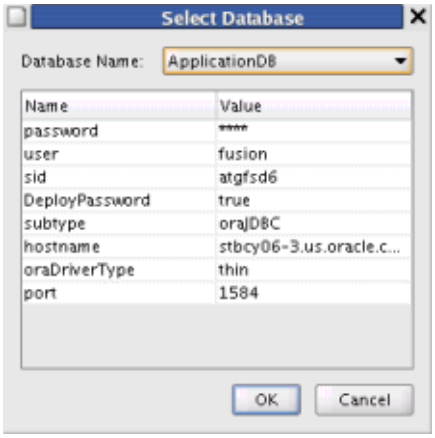
Figure 55–6 Launching the Seed Data Console



If there are multiple database connections in the workspace, the Seed Data Console – Select Database dialog, shown in Figure 55–7, displays.

Although this is not the normal case, if a developer needs to debug data on two different databases, this lets him or her choose which one to use.

Figure 55–7 Select Seed Data Database

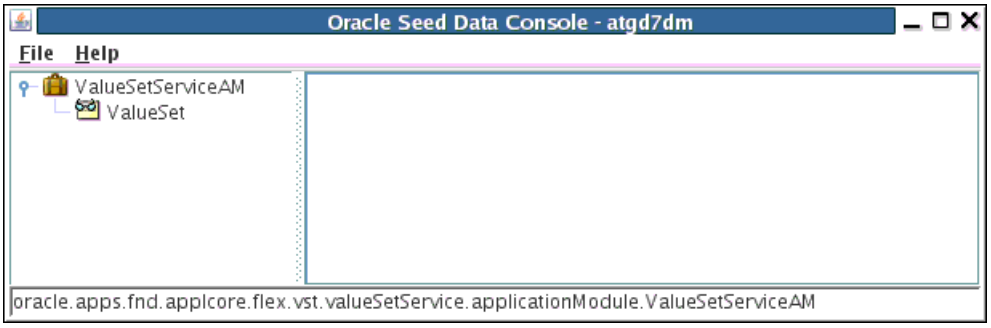


Select from the list of database connections available to the project. The default selected database is the default connection set on the Project Properties, under Business Components.

Click **OK** to launch the Seed Data Console with the selected connection information.

The Seed Data Console main page, as shown in Figure 55–8, displays.

Figure 55–8 Seed Data Console Main Page



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. As a result, users always will see a warning message indicating incremental uploads have been turned off. This is harmless and can be ignored.

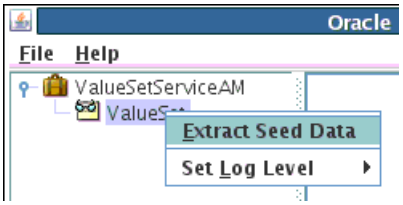
In the log, the warning will appear similar to:

```
[WARNING] 14:38:58 : -cfver parameter not passed,
incremental uploads turned off
```

55.2.4 How to Use Seed Data Extract Processing

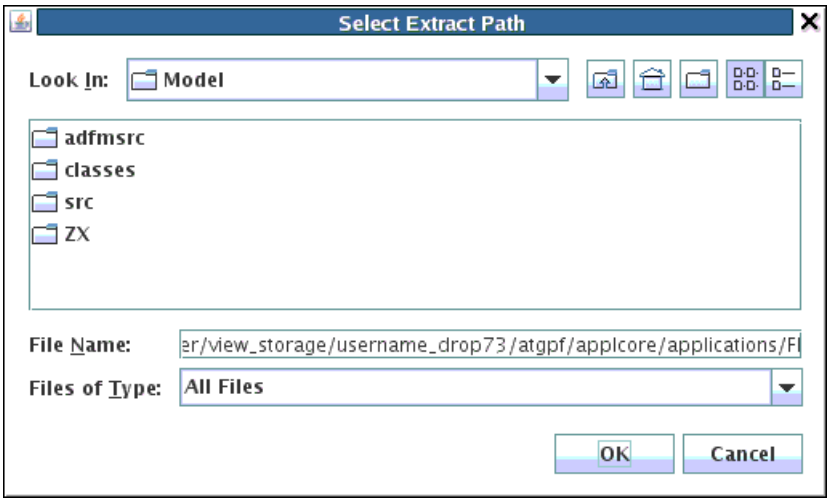
To start a Seed Data Extract operation, right-click the Driver view object from which you want to extract and select **Extract Seed Data** from the context menu, as shown in Figure 55–9.

Figure 55–9 Select Extract Seed Data item



The Select Extract Path dialog, shown in Figure 55–10, displays.

Figure 55–10 Select Seed Data extract path



Type a directory path location in the **File Name** text box, or browse to an existing location using the directory browser. The directory location need not necessarily exist when typing a new name, as the directory paths will be created as needed during the Extract process. The Extract directory path selected is used as the Seed Data Extract Root for the generated extract files.

55.2.4.1 Understanding Extract Taxonomy Partition Selection Dialogs

Seed Data Extract uses selection dialogs for Taxonomy partitions.

Functional Design

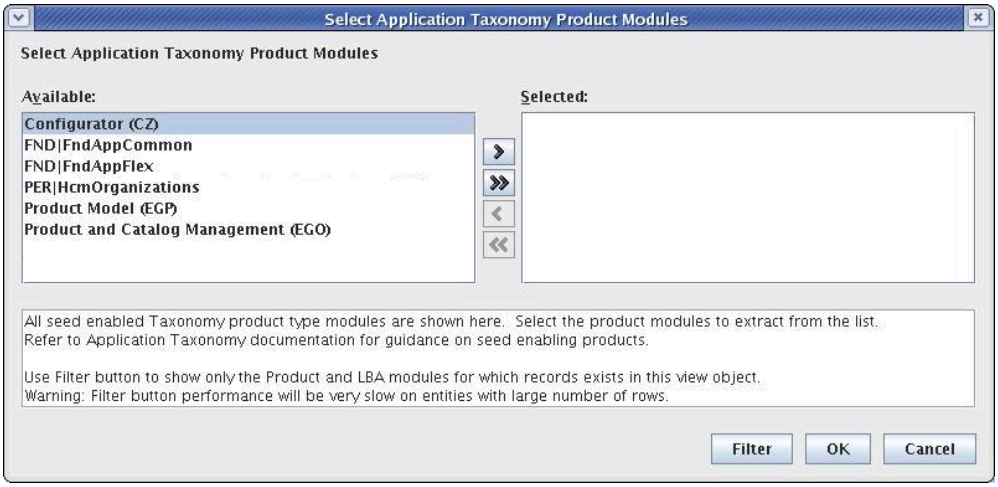
When starting the Extract process, if a Taxonomy Partition Attribute is found in the Driver view object, Extract will show the Taxonomy Partition selection dialogs. See "Determining the Taxonomy Partitioning Attribute" for steps taken by Extract to find the Taxonomy partition attribute, if any.

Taxonomy Products Dialog

The **Select Application Taxonomy Product Modules**, as shown in Figure 55–11, 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.

Figure 55–11 Select Application Taxonomy Product Modules dialog



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

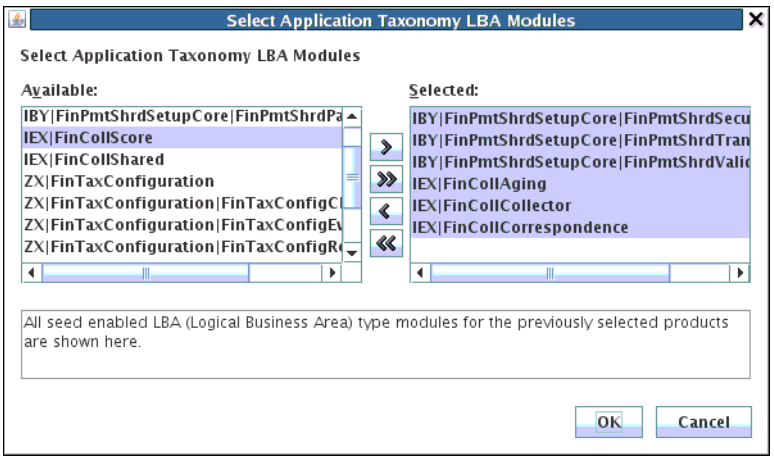
After selecting the Products, the second dialog, Select Application Taxonomy LBA Modules, may or may not be shown. If the selected Products were parents of explicitly available seed-enabled LBA modules, all the available LBAs for the selected Products are shown. Again, only LBA module types found in 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 55–12 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.

Figure 55–12 Selecting Application Taxonomy LBA Modules



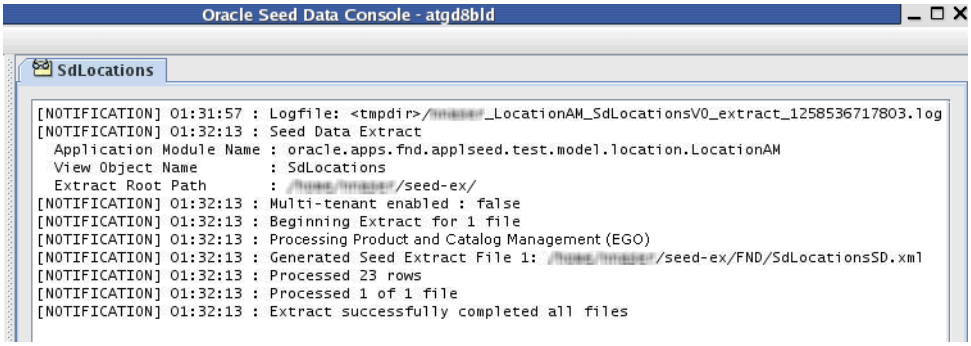
Extract Processing

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 55–13 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.

Figure 55–13 Example Output Selection



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 ApplTaxomomySeedDataPVO 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 Figure 55–1 would be added using adxml comments for the case where the current file is dependent on it.

Example 55–1 Sample Metadata Using ADXML Comments

```
<!-- <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 55–5.

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 55–3.

All files extracted from that view object would have the dependencies stamped in adxml comments.

Table 55–3 Application Module Custom Property for a View Object Instance

Application Module Custom Property Name	Value	Example
SD_DEPENDENT_FILES_<View object instance name>	<product>:<path>:<filename>, <product>:<path>:<filename>,.. The static dependent file location consists of three parts separated by colons: ■ **product**: Product short name. ■ **path**: Relative path from the product folder where the file exists. ■ **filename**: name of the seed xml file that contains the referenced data. Multiple static external references should be separated by a comma.	Property: SD_DEPENDENT_FILES_TimeDefinition1 Value: HCM:HCM/Per:LookupsSD.xml,FND:FND:ValueSetSD.xml

Turning Off Dynamic File Dependency Generation

You can turn off the dynamic file dependency generation by defining the custom property shown in Table 55–4 on the application module.

Table 55–4 Property to Turn Off Dynamic File Dependency

Application Module Custom Property Name	Value	Example
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.

55.2.4.2 Using the Extract Seed Data Command Line Interface

The Seed Data Extract process can also 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

```
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 <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

```
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 [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.

Seed Data Extract Command Line Parameters

The available Seed Data Extract parameters are listed in Table 55–5.

Table 55–5 Available Seed Data Parameters

Property	Value	Required?	Example
`dburl`	database connection URL in JDBC format without username and password	Yes	jdbc:oracle:thin:@stbcy06-4.us.oracle.com:1991:atgd7dm
`dbuser`	database user to be used for extract	Yes	fusion
`AM`	Application module name, fully package qualified	Yes	oracle.apps.fnd.applcore.flex.dff.descriptiveFlexfieldService.applicationModule.DescriptiveFlexfieldServiceAM
`VO`	Driver ViewObject instance name	Yes	DescriptiveFlexfield
`ExtractRootPath`	Path to extract seed data files	No	/home/seed/data
`PartitionKeyIds`	Comma delimited Taxonomy Id values to extract, either ModuleId or ApplicationId, depending on partition strategy.	No	Taxonomy Module Ids: 4F1F0DFC58F87DB4E04044981FC62F46, 47110F64AC8F08E2E040449823C60DB6 Application Ids: 250, 667, 10047
`PartitionKeyNames`	Comma delimited Taxonomy name values to extract, either Product codes, or LBA names, or combination thereof	No	FND, HCM, invoices, receivables, cashManagement

Table 55–5 (Cont.) Available Seed Data Parameters

Property	Value	Required?	Example
loglevel	The Log Level (SEVERE to FINEST).	No	-loglevel FINE
log	The location of the log file.	No	-log /home/seed/ex.log
Entid	Enterprise Id numeric value.	No	-entid 1

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 PartitionKeyIds or the PartitionKeyNames 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:

```
PartitionKeyNames = FND, HCM, #FCM, GL, AM
```

This will ignore the FCM value entry, but keep others intact. A sample `PartitionKeyNames` command line option is shown in Example 55–2.

Example 55–2 Sample Command Line

```
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.De
```

```
scriptiveFlexfieldServiceAM
-VO DescriptiveFlexfield
-ExtractRootPath /home/seed/data
-PartitionKeyNames HCM, invoices, cashManagement
-loglevel FINER
```

Example 55–3 shows the contents of the sample Seed Extract Command Property File, located at `/home/extract.properties`:

Example 55–3 Sample Command Property File

```
#Sample extract property file, comment line
AM =
oracle.apps.fnd.applcore.flex.dff.descriptiveFlexfieldService.applicationModule.De
scriptiveFlexfieldServiceAM
dburl = jdbc:oracle:thin:fusion/fusion@fully_qualified_server_name:1991:database_
name
dbuser = fusion
VO = DescriptiveFlexfield
ExtractRootPath = /home/seed/data
PartitionKeyNames = FND,HCM
```

Sample command line showing command line parameter to `/home/extract.properties` command property file.

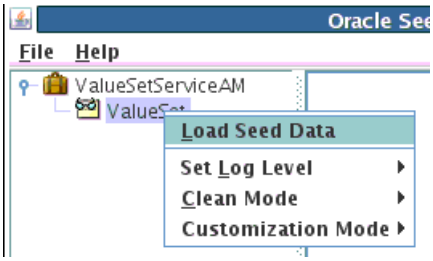
```
java -cp $jdev_
install/jdeveloper/jdev/oaext/lib/oracle.apps.fnd.applseed-rt.jar:$jdev_
install/oracle_common/modules/oracle.odl_11.1.1/ojdl.jar
oracle.apps.fnd.applseed.rt.extract.Extract /home/extract.properties
```

55.2.5 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 55–14 displays so you can select the **Load Seed Data** option.

Figure 55–14 Upload Seed Data Menu Option



In addition to the Load Seed Data option, discussed in Section 55.2.5.1, "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 is 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.

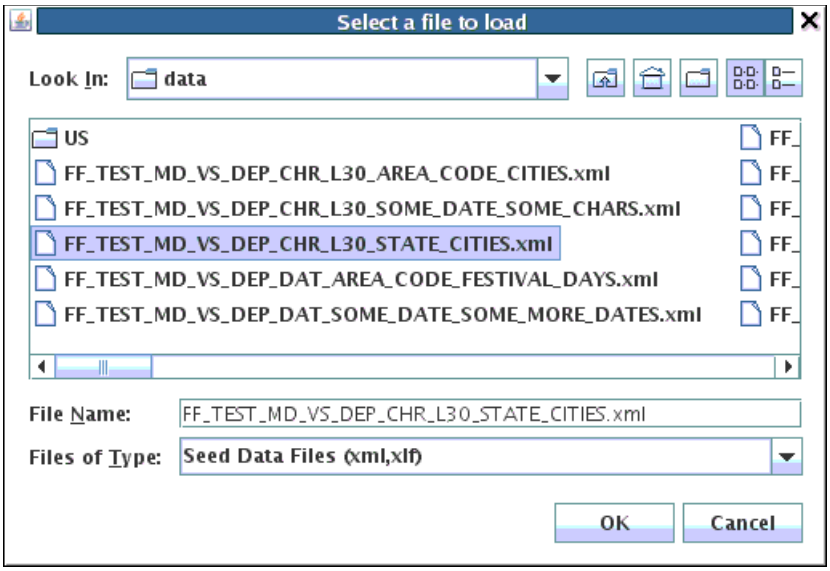
55.2.5.1 Uploading Seed Data

To upload Seed Data, right-click the desired Driver view object from the Consoleview object tree list. Select **Load Seed Data**.

> **Important:** 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 55–15, displays.

Figure 55–15 Selecting a File to Load



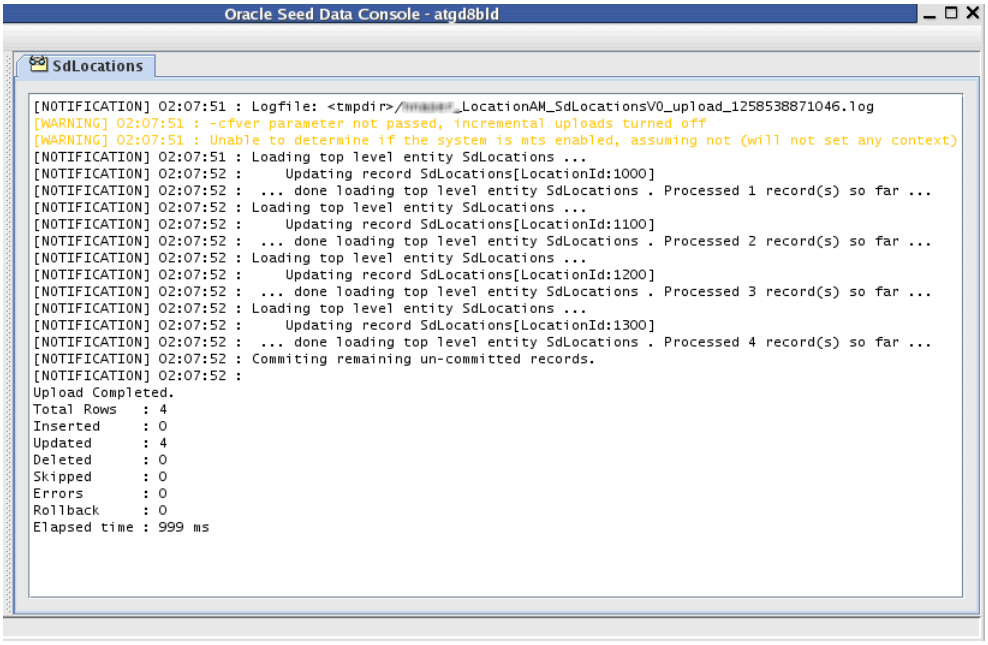
Select either an XML file or its corresponding translation xliff file to initiate a National Language Support (NLS) upload.

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 55–16.

Figure 55–16 Upload Output Messages



You may see warning messages about columns being not updateable. Review these messages to determine if you can ignore them for your specific case.

55.2.5.1.1 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 55–6, is set to a valid JDeveloper installation directory. Include the **jdeveloper** sub folder if you are using JDeveloper integrated with WebLogic Server.

Table 55–6 Environment Variables for the Seed Data Upload CLI

Variable name	Required?	Purpose
`JDEV_HOME`	No	Should point to the full path to the JDeveloper installation.
`APPLSEED_ 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 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.

`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. On the other hand, 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 55–4 shows a sample Seed Data upload command entered on the CLI, and Example 55–5 shows a sample command line command for a Key Flexfield application module.

Example 55–4 Sample Seed Data Upload Command on the CLI

```
java -cp $jdev_install/jdev/oaext/lib/oracle.apps.fnd.applseed-rt.jar
oracle.apps.fnd.applseed.rt.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
     [-cfver <checkfile version>]                                  // Obsolete. Do
not use.
     [-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
     [-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 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 55–5 Sample Command Line Invocation for a KFF Application Module

```
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.loader.Loader
```

55.2.5.1.2 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 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.

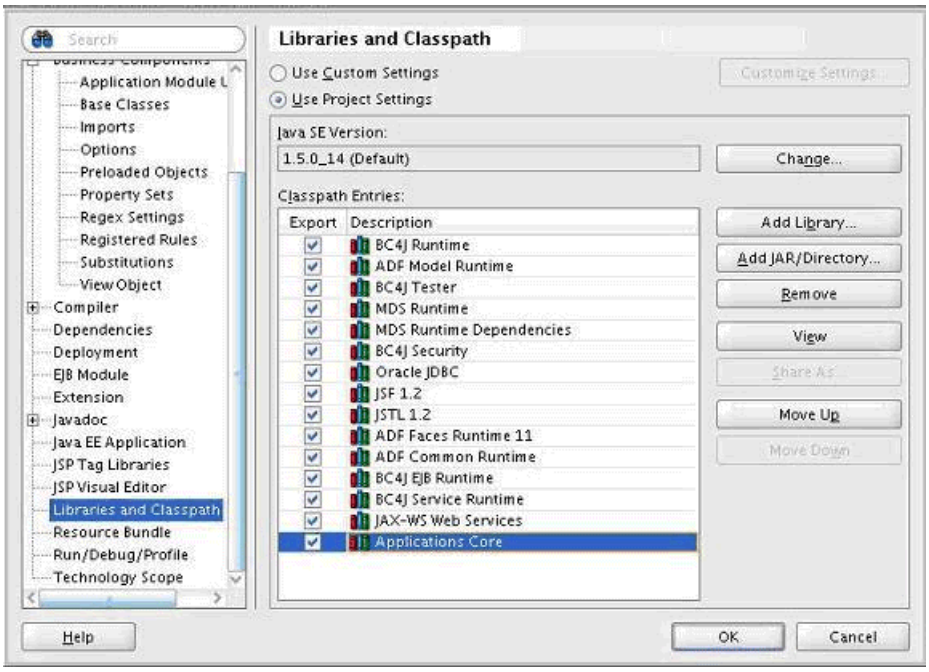
55.2.6 How to Share Application Modules

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 55–17, the team that is developing the application module packages the module into an ADF Business Components library.

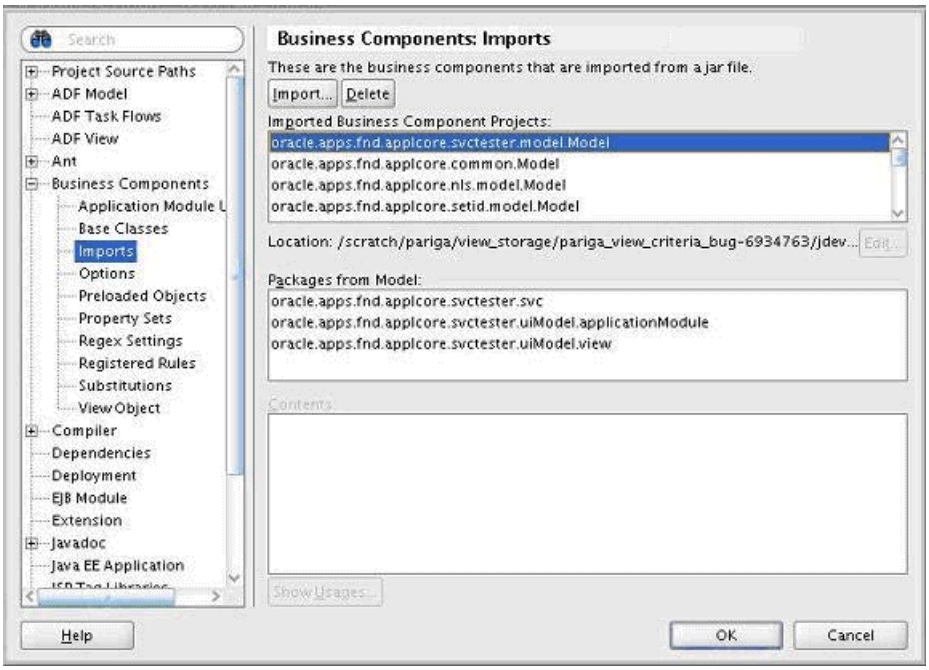
The consuming team includes these libraries in the project using the Libraries/Classpath feature of JDeveloper. Once imported, developers can use these libraries to extract and upload seed data, but cannot edit the Seed Data configuration.

Figure 55–17 Including Shared Libraries



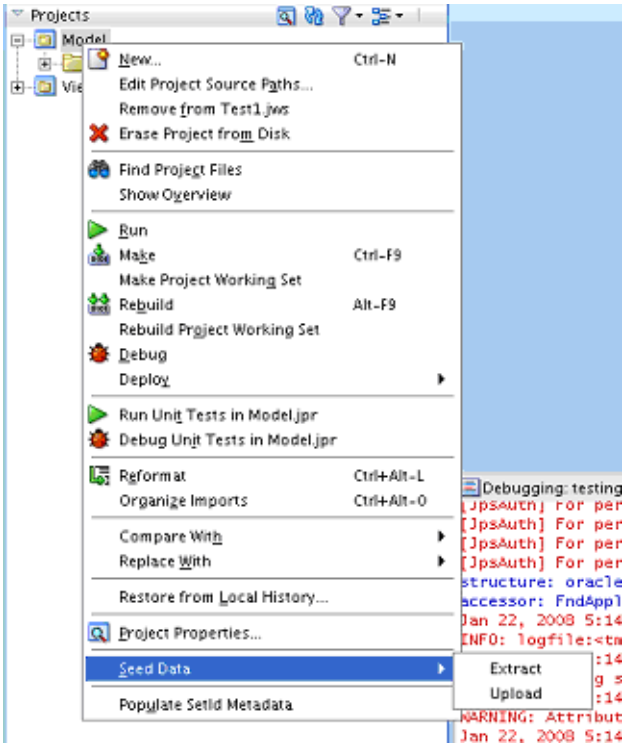
The imported Business Components can be seen using the Business Components Import feature of JDeveloper. Once imported, developers can use these libraries to extract and upload seed data, but cannot edit the Seed Data configuration.

Figure 55–18 Importing Business Components



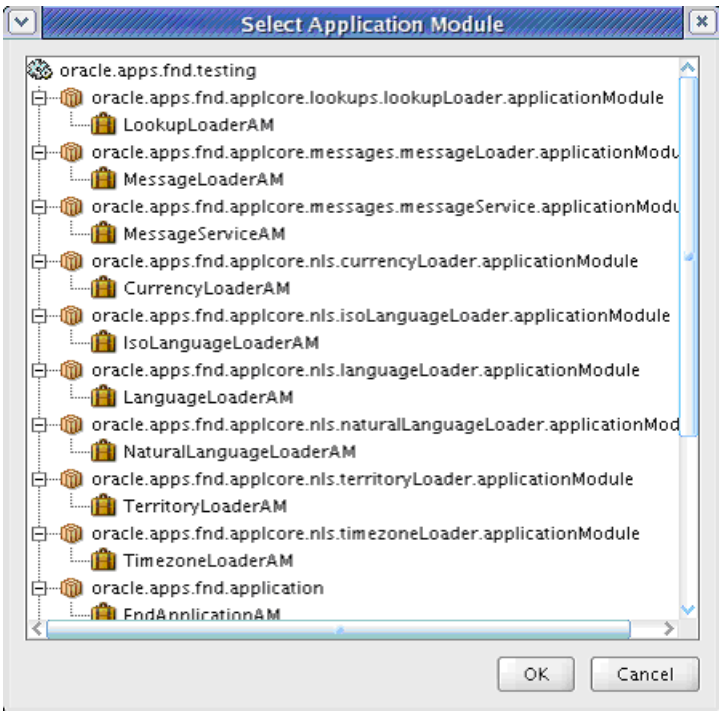
Since Application Modules shared using an ADF Business Components library are not shown by JDeveloper in the Application Navigator, users must right-click the Business Components project into which they imported the shared libraries. The Seed Data menus would also be available on any Business Components project node in the Applications Navigator tree of JDeveloper.

Figure 55–19 Accessing Seed Data from Business Components Project



Clicking any of the Seed Framework menu items will display a tree structure showing all the Seed Framework-enabled Application Modules (those that have at least one Seed Data driver view object configured within them) available to that project.

Figure 55–20 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.

55.2.7 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 55–7.

Table 55–7 Summary of Seed Data Update Features

Feature	Applicable to	Default Setting	Design time Control	Runtime Control
Incremental Updates	Only US language seed data files	Off	Enabled by setting SD_INCR_MIDTIER_<ViewDefinitionName> to true	Disabled by using optional command line parameter -noincr Disabled by setting APPLSEED_NO_INCREMENTAL to true
JDBC Mode	Only translation seed data files	On for newly extracted files	Disabled by setting SD_NLS_USE_ADF_<ViewDefinitionName> to true	Disabled by setting APPLSEED_NLS_USE_ADF to true

55.2.7.1 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. For certain kinds of seed data (such as large volumes of slowly changing data), it is desirable to be able to update

only those records from the seed data file that have changed since the file was last loaded on a given database. The Seed Data Framework provides an optional feature that allows product teams to incrementally update their seed data.

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 *only* to US language seed data XML files and is not enabled by default. For translated seed data, there is no provision to do incremental updates.

To enable 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 true to turn on incremental updates.

```
SD_INCR_MIDTIER_ValueSetVO true
```

Note that the incremental updates feature requires new metadata, in the form of checksums, to be embedded into the seed data file. As a result, 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`.

55.2.7.2 Implementing Java Database Connectivity-based National Language Support Updates

The Seed Data Loader loads seed data using the ADF components developed by product teams. As a result, 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, with the exception of 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 (XLF) 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`.

55.3 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.

55.3.1 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 XML file, with an extension of .xlf.

55.3.1.1 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 single unit. There is metadata in the files to ensure that the base and XLIFF files match and were extracted together.

55.3.2 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.

55.3.3 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 XML file must be successfully loaded prior to 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 XML file.

55.3.4 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 Section 9.2, "Using Multi-Language Support Features" for more information on creating translatable data models.

56

Using 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:** Prior to SXML migration, these were referred to as XDF (extension) files.

This chapter includes the following sections:

56.1 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.

56.2 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.

56.2.1 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.

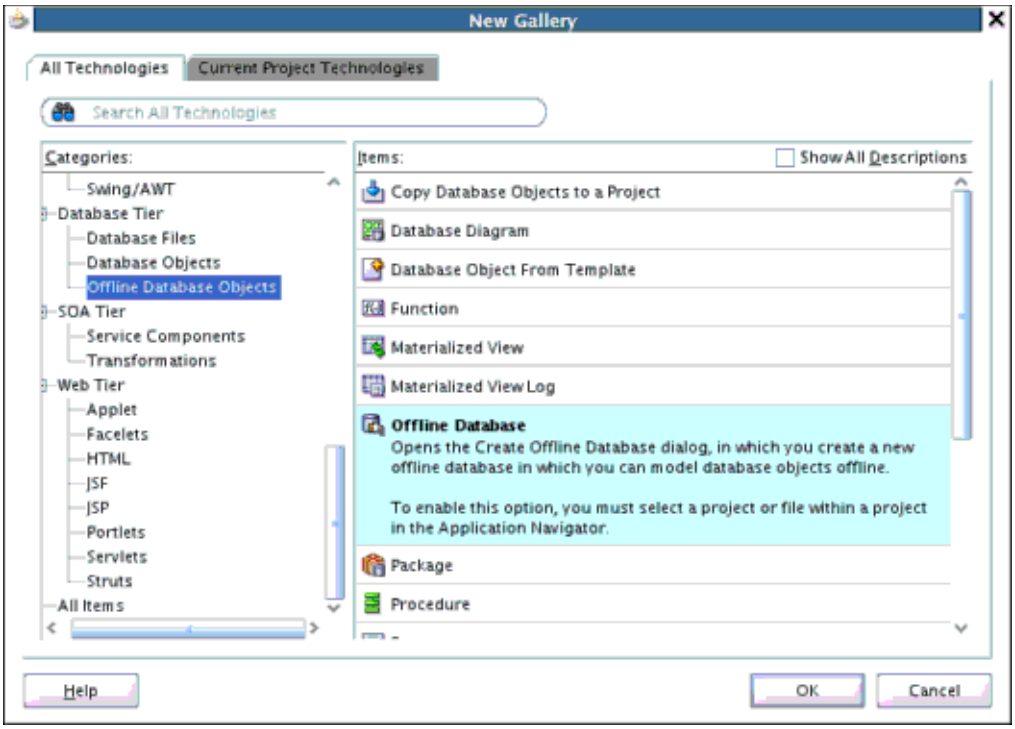
56.2.2 How to Create an Offline Database

Follow these directions to create an offline database.

To create an offline database:

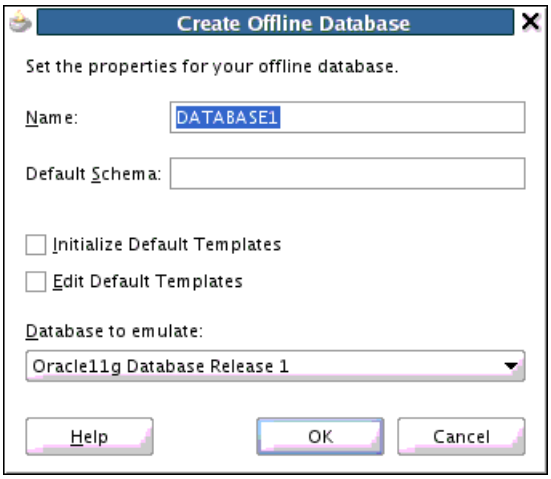
1. In the Application navigator within JDeveloper, locate the project you want to work in.
2. Select **File > New** to display the New Gallery.
3. From the New Gallery, select **Database Tier > Offline Database Objects > Offline Database**, as shown in Figure 56–1.

Figure 56–1 Creating a New Offline Database



4. In the Create Offline Database dialog, enter a name for the offline database, as shown in Figure 56–2. For more information at any time, press **F1** or click Help from within the Create Offline Database dialog.

Figure 56–2 Naming the New Offline Database



The following types of objects are modeled using offline database objects.

- Table
- Trigger
- View
- Materialized View
- Materialized View Log

- Sequence
- Synonym

JDeveloper offline database objects do not support these objects. However, SXML persistence files for these object types can be imported using the applxdf extension.

- Queue
- Queue tables
- Policy

56.2.3 How to Deploy an Offline Database in XML Persistence Format

See Section 56.2.9.1, "Deploying in SXML Persistence Format."

56.2.4 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:

```
Table.TYPE
   View.TYPE
   Synonym.TYPE
Sequence.TYPE
MaterializedViewLog.TYPE
MaterializedView.TYPE
Trigger.TYPE
```

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.

56.2.5 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 Views
- User Defined Properties for Sequence
- User Defined Properties for Materialized View
- User Defined Properties for Materialized View Log
- User Defined Properties for Trigger

56.2.5.1 User Defined Properties for Tables

Table 56–1 shows the User Defined Properties that are defined for the tables.

Table 56–1 User Defined Properties for Tables

UDP	Display Name	Values in JDeveloper	Definition in FND_TABLES
`isFlashbackAllowed` This property indicates whether flashback of the table is allowed. This UDP is mandatory.	Is Flashback Allowed	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.	FLASHBACK_ALLOWED
`isLogical` The value of this UDP indicates whether the deployment program will create an editioning view for this table or not. This UDP is required.	Editioning View	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.	LOGICAL VARCHAR2(1) Not Null
`runTwice`	Run Deployment Twice	Y: Specifies that the table needs to 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 a 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).	N/A
`shortName` 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.	Table Short Name	N/A 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.	SHORT_NAME VARCHAR2(30) Null
`objectOwner` This UDP stores the Short Name of the application that the table belongs to. This UDP is required.	Table Owner	N/A	APPLICATION_SHORT_NAME VARCHAR2(10)

Table 56–1 (Cont.) User Defined Properties for Tables

UDP	Display Name	Values in JDeveloper	Definition in FND_TABLES
`tsClassification` This UDP stores the Tablespace Classification for this table. This value is used to derive the tablespace for the table but it is not equivalent to the tablespace. This UDP is mandatory.	Tablespace Classification	TRANSACTION TABLES (default) REFERENCE INTERFACE SUMMARY NOLOGGING TRANSACTION_INDEXES ARCHIVE TOOLS MEDIA	N/A
`mlsSupportModel` The value of this UDP indicates the language data model for the table. This UDP is required.	MLS Support Model	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.	MLS_SUPPORT_MODEL VARCHAR2(30)
`status` The value of this UDP indicates the status of the table. This UDP is required.	Status	Active (default): The table is active. Obsolete: The table is obsolete and can be deleted from the database.	STATUS VARCHAR2(30)
`extensionOfTable` This UDP stores the name of the base table that is extended by this table.	Extension of Table	N/A	EXTENSION_OF_TABLE VARCHAR2(30)
`deployTo` The value of this UDP indicates the deployment mode of the table for the Oracle Fusion Disconnected Mobile Framework. This UDP is required.	Deploy To	Server DB Only (default): The table is deployed on the server database but not on the mobile database. All: The table is deployed both on the server database and on the mobile database. Mobile DB Only: The table is deployed on the mobile database but not on the server database.	DEPLOY_TO VARCHAR2(30)

Table 56–1 (Cont.) User Defined Properties for Tables

UDP	Display Name	Values in JDeveloper	Definition in FND_TABLES
`conflictResolution` The value of this UDP indicates how the Oracle Fusion Disconnected Mobile Framework should resolve the conflicts about duplicate rows. This UDP is required.	Conflict Resolution	Duplicate (default): A new duplicate record is added and the conflict will be handled during the next synchronization. This value should be used for non-intersection tables. Merge: The records are merged. This value should be used for intersection tables.	CONFLICT_RESOLUTION VARCHAR2(30)
`sharedObject` The value of this UDP indicates whether the table is accessed by external products.	Shared Object	Y: The table can be accessed directly by external products, other than the owning product. N (default): The table cannot be accessed directly by external products.	SHARED_OBJECT VARCHAR2(30)
`adxml` The value of this UDP is patch metadata used by the patching tool.	ADXML	N/A	N/A
`axdmlFk` The value of this UDP is patch metadata used by the foreign key portion of the patching tool.	ADXML for Foreign Keys	N/A	N/A
`adxmlDeferredIndexes` The value of this UDP is patch metadata used by the deferred indexes portion of the patching tool.	ADXML for Deferred Indexes	N/A	N/A
`useExistingAdxml` This UDP is mandatory.	Use Existing ADXML	Y: Generate ADXML comment using the existing ADXML value from the ADXML UDP N (default): Regenerate the ADXML comment and update the ADXML UDP.	N/A
`isSelectAllowed` This property indicates whether the select on the table is allowed. This is mandatory.	Is Select Allowed	Y (default) N	SELECT_ALLOWED VARCHAR2(1) Not Null
`isUpdateAllowed` This property indicates whether update on the table is allowed. This property is mandatory.	Is Update Allowed	Y (default) N	UPDATE_ALLOWED VARCHAR2(1) Not Null
`isInsertAllowed` This property indicates whether the insert on the table is allowed. This property is mandatory.	Is Insert Allowed	Y N (default)	INSERT_ALLOWED VARCHAR2(1) Not Null

Table 56–1 (Cont.) User Defined Properties for Tables

UDP	Display Name	Values in JDeveloper	Definition in FND_TABLES
`isDeleteAllowed` This property indicates whether delete on the table is allowed. This property is mandatory.	Is Delete Allowed	Y (default) N	DELETE_ALLOWED VARCHAR2(1) Not Null
`isTruncateAllowed` This property indicates whether truncate on the table is allowed. This property is mandatory.	Is Truncate Allowed	Y N (default)	TRUNCATE_ALLOWED VARCHAR2(1) Not Null
`maintainPartition` Indicates whether partitions can be maintained on the table. This property is mandatory.	Maintain Partition	Y N (default)	MAINTAIN_PARTITION VARCHAR2(1) Not Null
`exchangePartition` Indicates whether it is possible to exchange partitions on the table. This property is mandatory.	Exchange Partition	Y N (default)	EXCHANGE_PARTITION VARCHAR2(1) Not Null
`maintainIndex` This property indicates whether it is possible to maintain indexes on the table. This property is mandatory.	Maintain Index	Y N (default)	MAINTAIN_INDEX VARCHAR2(1) Not Null

56.2.5.2 User Defined Properties for Columns

Table 56–2 shows the User Defined Properties that are defined for columns.

Table 56–2 User Defined Properties for Columns

UDP	Display Name	Values in JDeveloper	Definition in FND_COLUMNS
`shortName` 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.	Column Short Name	N/A	SHORT_NAME VARCHAR2(27) Null
`translateFlag` The value of this UDP indicates whether the column is translatable or not. This UDP is required.	Translate	Y: The column is translatable. N (default): The column is not translatable.	TRANSLATE_FLAG VARCHAR2(1) Not Null
`status` The value of this UDP indicates the status of the column. This UDP is required.	Status	Active (default): The column is active. Obsolete: The column is obsolete and can be deleted from the database.	STATUS VARCHAR2(30)
`customDefaultValue`	Custom Default Value	N/A	N/A
`denormPath` 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.	Denormalization Path	N/A	N/A
`routingMode` The value of this UDP indicates how this column will be handled during the synchronization between the server and the client database as per the Oracle Fusion Disconnected Mobile Framework.	Routing Mode	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.	ROUTING_MODE VARCHAR2(30)

Table 56–2 (Cont.) User Defined Properties for Columns

UDP	Display Name	Values in JDeveloper	Definition in FND_COLUMNS
`histogram` The value of this UDP indicates if the column is a candidate for histogram. This UDP is required.	Histogram	Y (default): This column is a candidate for histogram. N: This column is not a candidate for histogram.	N/A
`histogramSize` The value of this UDP indicates the number of buckets to be used when the column is defined as a candidate for histograms.	Histogram Size		N/A
`versionColumn` The value of this UDP indicates the name of the version column used by the Oracle Fusion Disconnected Mobile Framework during synchronization of LOB columns.	Disconnected Mobile Version Column Name	N/A	VERSION_COLUMN VARCHAR2(30 CHAR)

56.2.5.3 User Defined Properties for Indexes

Table 56–3 shows the User Defined Properties that are defined for the indexes.

Table 56–3 User Defined Properties for Indexes

UDP	Display Name	Values in JDeveloper	Definition in FND_INDEXES
`shortName` The short name of the index is used by the Zero Down Time patching programs to uniquely identify the index. The value of this UDP is defaulted to the index name when the length of the index name is less than 28 characters. When the length of the index name is greater than 28 characters, the developer must enter a value that uniquely identifies the index.	Index Short Name	N/A	SHORT_NAME VARCHAR2(30) Null
`deferred` The value of this UDP indicates whether the creation of the index will be deferred during deployment. This UDP is required.	Index Deferred (Y/N)	N (default): The creation of the index will not be deferred. Y: The creation of the index will be deferred.	N/A
`status` The value of this UDP indicates the status of the index. This UDP is required.	Status	Active (default): The index is active. Obsolete: The index is obsolete and can be deleted from the database.	STATUS VARCHAR2(30)
`deployTo` The value of this UDP indicates the deployment mode of the index for the Oracle Fusion Disconnected Mobile Framework. This UDP is required.	Deploy To	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.	DEPLOY_TO VARCHAR2(30)

56.2.5.4 User Defined Properties for Constraints

Table 56–4 shows the User Defined Properties that are defined for the constraints.

Table 56–4 User Defined Properties for Constraints

UDP	Display Name	Values in JDeveloper	Definition in FND_PRIMARY_KEYS or FND_FOREIGN_KEYS
`isLogical`	Logical Constraint	Y (default) N	LOGICAL VARCHAR2(1)
`shortName` 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.	Constraint Short Name		SHORT_NAME VARCHAR(30)
`conDefer` The value of this UDP indicates whether the creation of the constraint will be deferred during deployment. This UDP is required.	Defer Constraint	N (default): The creation of the constraint will not be deferred. Y: The creation of the constraint will be deferred.	N/A
`status` The value of this UDP indicates the status of the constraint. This UDP is required.	Status	Active (default): The constraint is active. Obsolete: The constraint is obsolete and can be deleted from the database.	N/A

56.2.5.5 User Defined Properties for Views

Table 56–5 shows the User Defined Properties that are defined for the views.

Table 56–5 User Defined Properties for Views

UDP	Display Name	Values in JDeveloper	Definition in FND_VIEWS
`adxml` The value of this UDP is patch metadata used by the patching tool.	ADXML	N/A	N/A
`isFlashbackAllowed` This property indicates whether flashback of the view is allowed. This UDP is mandatory.	Is Flashback Allowed	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.	FLASHBACK_ALLOWED
useExistingAdxml	Use Existing ADXML	Y: Generate ADXML comment using existing ADXML value from ADXML UDP. N (default): Regenerate ADXML comment and update ADXML UDP.	N/A
`status` The value of this UDP indicates the status of the view. This UDP is required.	Status	Active (default): The view is active. Obsolete: The view is obsolete and can be deleted from the database.	STATUS VARCHAR2(30)
`isSelectAllowed` This property indicates whether the select on the table is allowed. This is mandatory.	Is Select Allowed	Y (default) N	SELECT_ALLOWED VARCHAR2(1) Not Null
`isUpdateAllowed` This property indicates whether update on the table is allowed. This property is mandatory.	Is Update Allowed	Y (default) N	UPDATE_ALLOWED VARCHAR2(1) Not Null
`isInsertAllowed` This property indicates whether the insert on the table is allowed. This property is mandatory.	Is Insert Allowed	Y N (default)	INSERT_ALLOWED VARCHAR2(1) Not Null
`isDeleteAllowed` This property indicates whether delete on the table is allowed. This property is mandatory.	Is Delete Allowed	Y (default) N	DELETE_ALLOWED VARCHAR2(1) Not Null

56.2.5.6 User Defined Properties for Sequence

Table 56–6 shows the User Defined Properties that are defined for Sequence.

Table 56–6 User Defined Properties for Sequence

UDP	Display Name	Values in JDeveloper	Definition in FND_SEQUENCES
`objectOwner`	Sequence Owner	N/A	N/A
`status` The value of this UDP indicates the status of the sequence. This UDP is required.	Status	Active (default): The sequence is active. Obsolete: The sequence is obsolete and can be deleted from the database.	STATUS VARCHAR2(30) Null
`adxml` The value of this UDP is patch metadata used by the patching tool.	ADXML	N/A	N/A
`useExistingAdxml`	Use Existing ADXML	Y: Generate ADXML comment using existing ADXML value from ADXML UDP. N (default): Regenerate ADXML comment and update ADXML UDP.	N/A
`isSelectAllowed` This property indicates whether select on the table is allowed. This UDP is mandatory.	Is Select Allowed	Y (default) N	SELECT_ALLOWED VARCHAR2(1) Not Null
`resetSequence` This property indicates if the sequence can be reset to a specific value. This UDP is mandatory.	Reset Sequence	Y N (default)	RESET_SEQUENCE VARCHAR2(1) Not Null

56.2.5.7 User Defined Properties for Materialized View

Table 56–7 shows the User Defined Properties that are defined for the Materialized View.

Table 56–7 User Defined Properties for Materialized View

UDP	Display Name	Values in JDeveloper	Definition in FND_MVIEWS
`objectOwner` Short Name of the application to which this materialized view belongs.	Mview Owner	N/A	N/A
`shortName` Materialized view short name. Max Length is 24.	Materialized View Short Name	N/A	SHORT_NAME VARCHAR2(30)
`status` The value of this UDP indicates the status of the materialized view.	Status	Active (default): The materialized view is active. Obsolete: The materialized view is obsolete and can be deleted from the database.	STATUS VARCHAR2(30)

Table 56–7 (Cont.) User Defined Properties for Materialized View

UDP	Display Name	Values in JDeveloper	Definition in FND_MVIEWS
`adxml` The value of this UDP is patch metadata used by the patching tool.	ADXML		N/A
`tsClassification`	Tablespace Classification	TRANSACTION_TABLES REFERENCE INTERFACE SUMMARY (default) ARCHIVE TOOLS MEDIA	N/A
`useExistingAdxml` This UDP is mandatory.	Use Existing ADXML	Y: Generate ADXML comment using existing ADXML value from ADXML UDP. N (default): Regenerate ADXML comment and update ADXML UDP.	N/A
`isSelectAllowed` This property indicates whether select on the table is allowed. This UDP is mandatory.	Is Select Allowed	Y (default) N	SELECT_ALLOWED VARCHAR2(1) Not Null
`adxmlDeferredIndexes` The value of this UDP is patch metadata used by the deferred indexes portion of the patching tool.	ADXML for Deferred Indexes	N/A	N/A

56.2.5.8 User Defined Properties for Materialized View Log

Table 56–8 shows the User Defined Properties that are defined for the Materialized View Log.

Table 56–8 User Defined Properties for Materialized View Log

UDP	Display Name	Values in JDeveloper
`status` The value of this UDP indicates the status of the materialized view log. This UDP is required.	Status	Active (default): The materialized view log is active. Obsolete: The materialized view log is obsolete and can be deleted from the database.

Table 56–8 (Cont.) User Defined Properties for Materialized View Log

UDP	Display Name	Values in JDeveloper
`objectOwner` Materialized view log owner.	MV Log Owner	N/A
`adxml` The value of this UDP is patch metadata used by the patching tool.	ADXML	
`useExistingAdxml` This UDP is mandatory.	Y N (default)	Y: Generate ADXML comment using existing ADXML value from ADXML UDP. N (default): Regenerate ADXML comment and update ADXML UDP.

56.2.5.9 User Defined Properties for Trigger

Table 56–9 shows the User Defined Properties that are defined for Trigger.

Table 56–9 User Defined Properties for Trigger

UDP	Display Name	Values in JDeveloper
`status` The value of this UDP indicates the status of the trigger. This UDP is required.	Status	Active (default): The trigger is active. Obsolete: The trigger is obsolete and can be deleted from the database.
`objectOwner`	Trigger Owner	N/A
`adxml` The value of this UDP is patch metadata used by the patching tool.	ADXML	
`useExistingAdxml` This UDP is mandatory.	Use Existing ADXML	Y: Generate ADXML comment using existing ADXML value from ADXML UDP. N (default): Regenerate ADXML comment and update ADXML UDP.

56.2.6 How to Create an Offline Database Object

To create new offline database objects from within JDeveloper, in the Application Navigator:

- Right-click the offline Database or schema.
- Select **New Database Object**.
- 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.

56.2.7 How to Edit an Offline Database Object

To edit an offline database object:

- In the Application Navigator, expand the workspace, project, and schema.

- 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.

- 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.

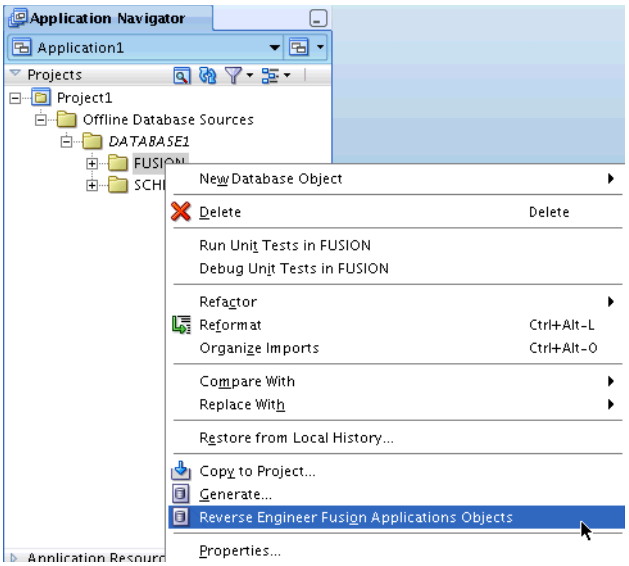
56.2.8 How to Import an Offline Database Object

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

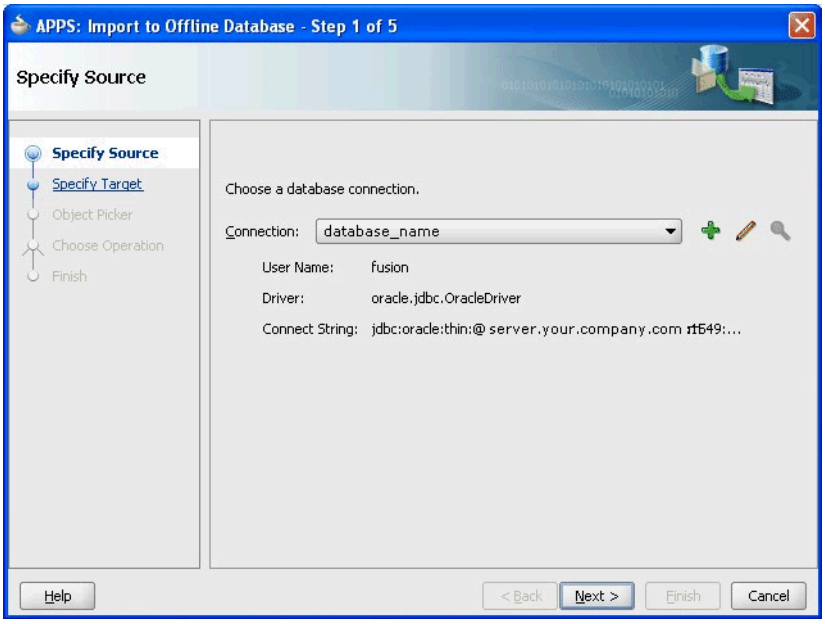
Object definitions can be generated to the Offline Database by right-clicking an Offline Database Source and selecting the **Reverse Engineer Fusion Applications Objects** option, extended to invoke the relevant import API, as shown in Figure 56–3.

Figure 56–3 Starting the Import Database Object Wizard



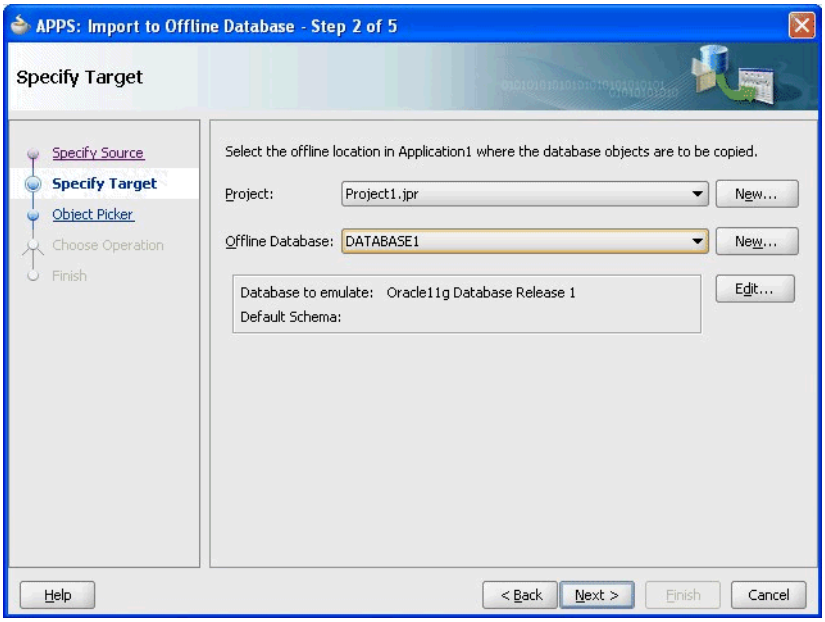
1. The target database connection name can be selected from the list of all defined database connections, or a new connection can be created in the **Specify Source** dialog, shown in Figure 56–4.

Figure 56–4 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 56–5.

Figure 56–5 Specifying the Target



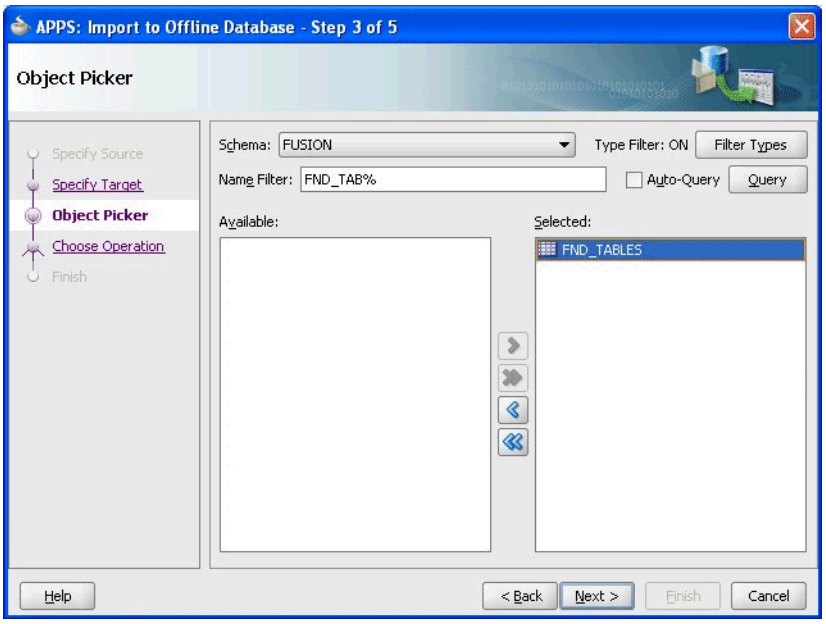
Filters can be applied to select the objects that are displayed as available for import. When there are a large number of objects in the schema, you should apply filters.

In the Object Picker, shown in Figure 56–6, 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** once you have entered the filter you want to use.
- Select the object types you want to view.

Figure 56–6 Picking an Object



3. Click **Next** to display the summary information.
4. Click **Finish** to import the selected objects to the specified offline database.

56.2.9 How to Deploy the Offline Database Objects

Once an offline database object is created, it can be deployed to a target database using the deployment extension provided in JDeveloper.

56.2.9.1 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:

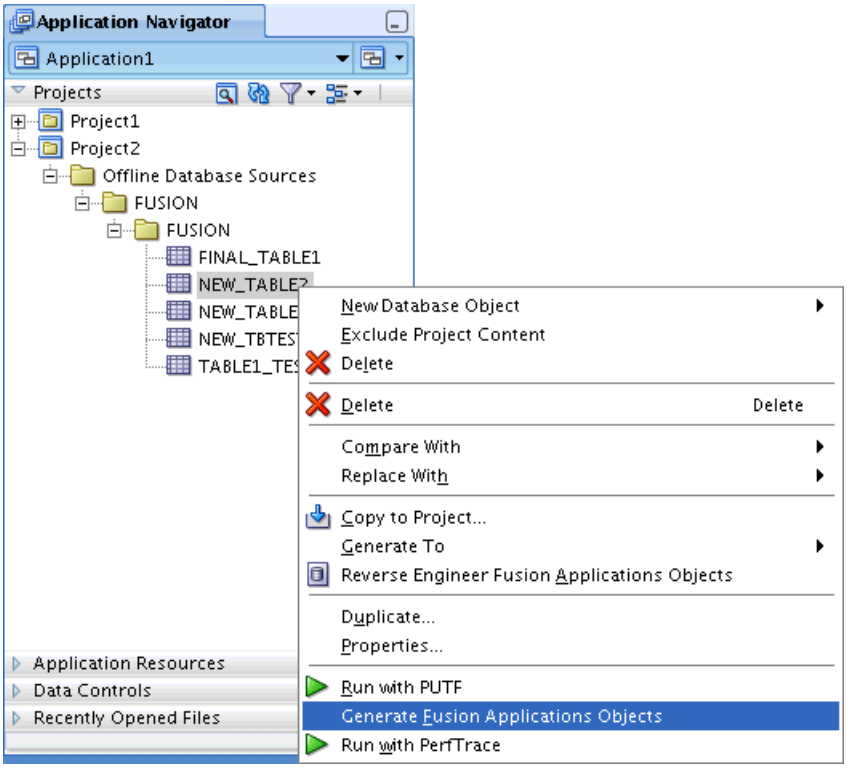
- Deployment wizard extension that can be invoked from within JDeveloper.
- Standalone deployment program that can be invoked from the command line.

56.2.9.1.1 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:

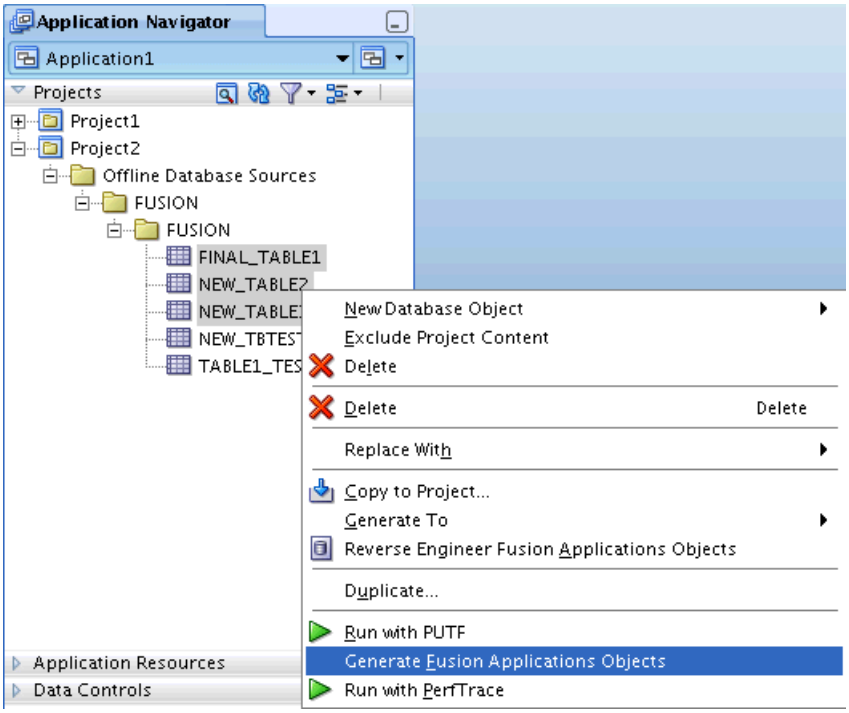
- Deploying a single database object file. Only one item is selected, as shown in Figure 56–7.

Figure 56–7 Deploying a Single Database Object File



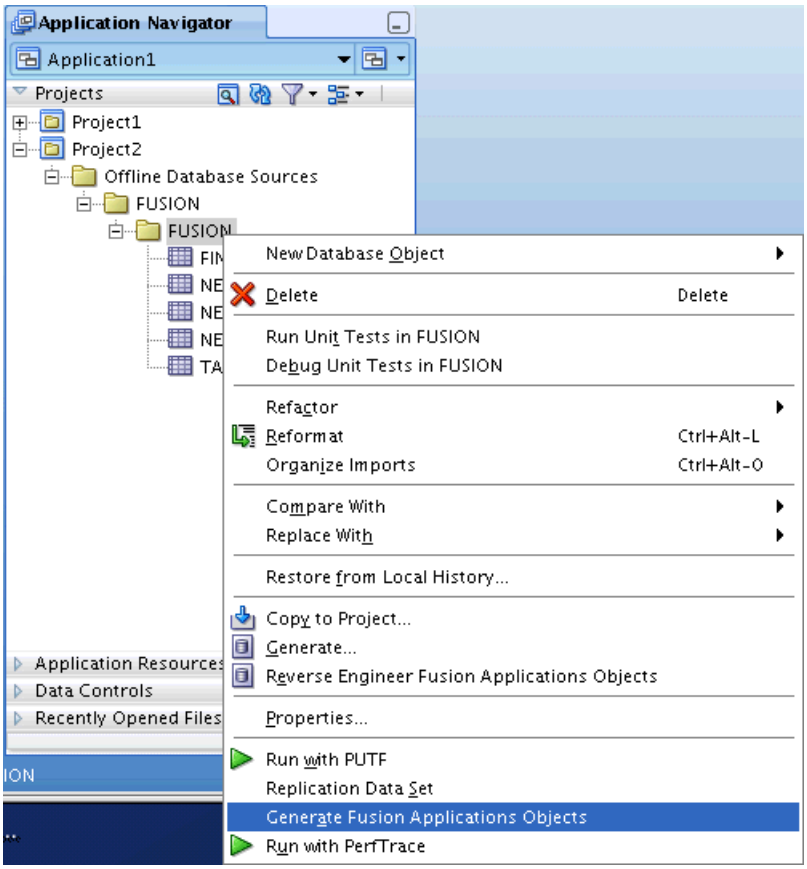
- Deploying multiple database object files (Bulk Deployment). Several items are selected, as shown in Figure 56–8.

Figure 56–8 Deploying Multiple Database Object Files



- Deploying offline schema object (and thereby deploying the entire designed data model), as shown in Figure 56–9.

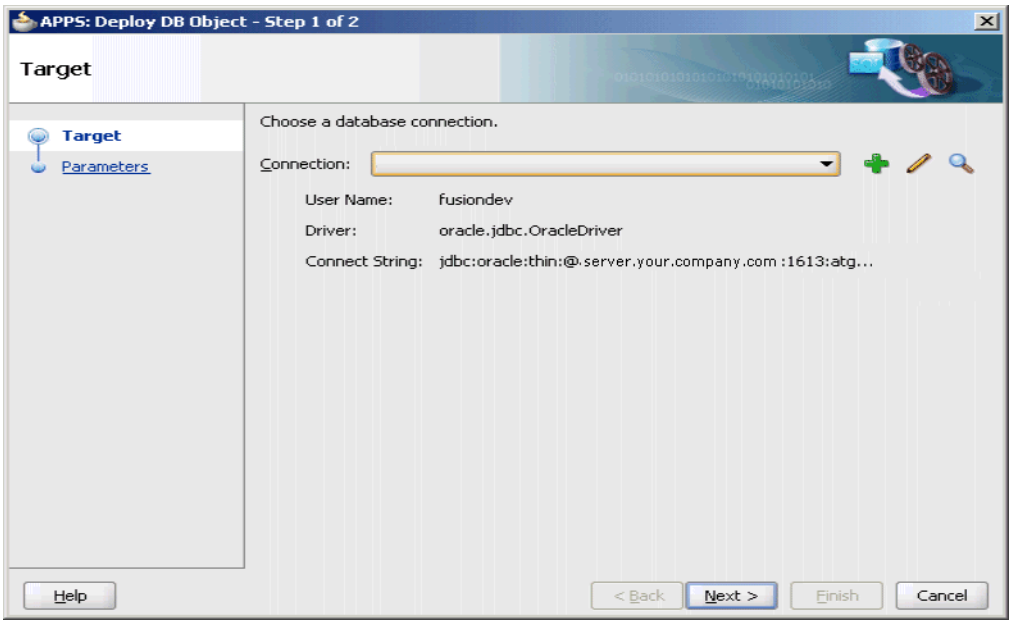
Figure 56–9 Deploying Offline Schema Object



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 56–10.

Figure 56–10 Selecting a Database Connection



- Deployment Parameters. Figure 56–11 shows how the dialog appears for single database deployment parameters.

Figure 56–11 Single Database Object Deployment Parameters

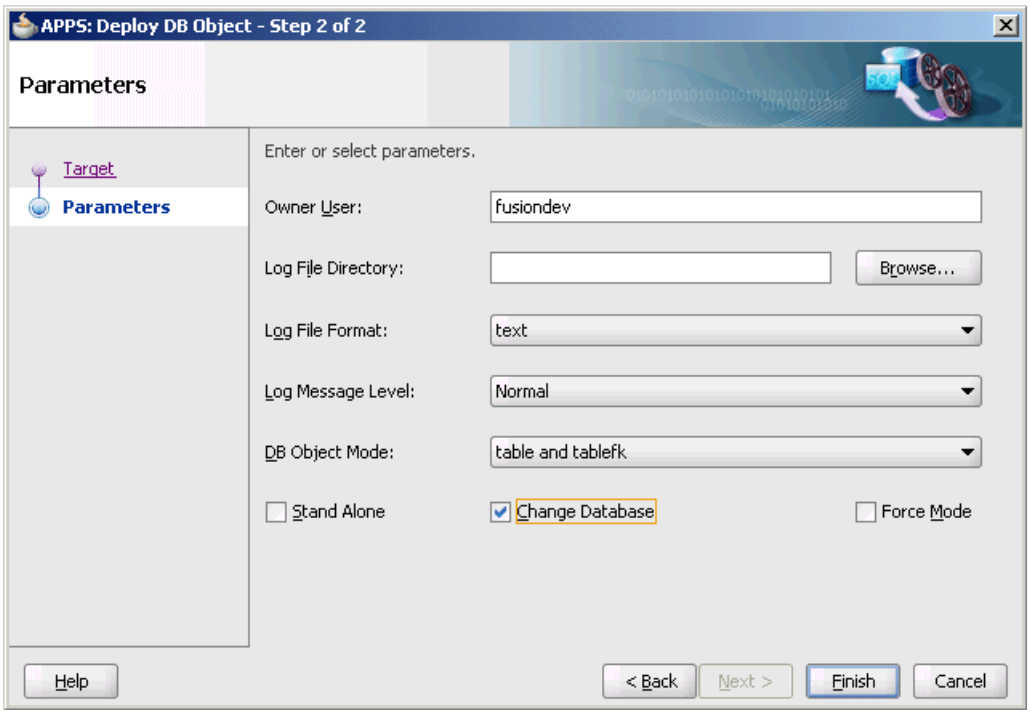
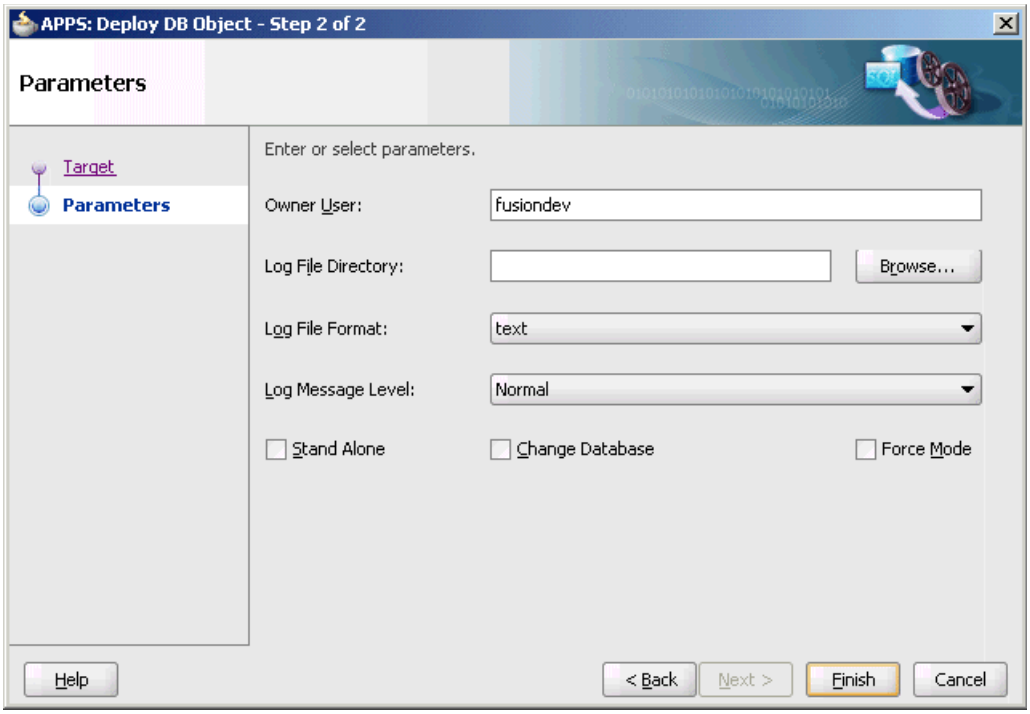


Figure 56–12 shows how the Deployment Parameters dialog appears for multiple database deployment parameters.

Figure 56–12 Multiple Database Object 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.

56.2.9.1.2 How to Use the Database Object Deployment Command Line Interface

Use this command and parameters shown in Example 56–1 to deploy a database object from the command line.

Example 56–1 Sample Database Object Deployment Using the CLI

```
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}]*
```

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."

56.2.9.2 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:

- ojdl.jar (Only DROP8 Build 3)
- ojdl2.jar (Only DROP8 Build 3)
- xmlparserv2.jar
- ojdbc6.jar
- orai18n.jar
- oracle.apps.fnd.applxdf.jar

Example 56–2 shows the set of commands to set the CLASSPATH.

Example 56–2 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/wlserver_
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
```

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.

56.2.9.3 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 56–3.

Example 56–3 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>
```

56.2.9.4 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 Section 56.2.9.5.3, "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 or not data needs to be preserved or migrated. Once 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.

56.2.9.5 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.

56.2.9.5.1 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.

56.2.9.5.2 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** needs to be used to synchronize the XDF dictionary.

56.2.9.5.3 How to Use fnd_cleanup_pkg and fnd_drop_obsolete_objects Use fnd_cleanup_pkg and fnd_drop_obsolete_objects to clean a database.

Using fnd_cleanup_pkg

Procedure fndcleanup(name): Remove table/sequence/view names that are in fnd_tables/fnd_views/fnd_sequences tables, but not in the database. All the required XDF dictionary tables will be updated when fnd_tables/fnd_views/fnd_sequences is updated.

name: Optional, if it is not defined. Remove all table/view/sequence names that are in fnd_tables/fnd_views/fnd_sequence tables, but not in the database.

Examples

- Remove all table/view/sequence information in fnd_tables/fnd_views/fnd_sequences tables, but not in the database. All the required XDF dictionary tables will be updated when fnd_tables/fnd_views/fnd_sequences is updated.

  ```
  execute fnd_cleanup_pkg.fndcleanup
  ```

- 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/fnd_views tables that do not exist in the database

  ```
  execute fnd_cleanup_pkg.fndcleanup('HZ%')
  ```

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 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/views with name like 'HZ_%" for dropping tables/views or drop columns if tables are not obsolete.

  ```
  execute fnd_drop_obsolete_objects.drop_object('HZ_%')
  ```

56.2.9.5.4 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:

      ```
      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:

      ```
      execute fnd_drop_obsolete_objects.drop_object(' obj')
      ```
    - Execute this command from sqlplus:

      ```
      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.

  ```
  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.

  ```
  execute fnd_cleanup_pkg.fndcleanup
  ```
- How do I remove obsolete columns/indexes in objects from FND data?

  Execute this command from sqlplus.

  ```
  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.

  ```
  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.

  ```
  execute fnd_drop_obsolete_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.

  ```
  execute fnd_drop_obsolete_objects.drop_object('HZ%')
  ```

56.3 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 56–10, Table 56–11, Table 56–12, and Table 56–13 present the user defined properties that will be defined for each object type.

Table 56–10 Table Object Type Properties

UDP Name	Description	Values
`Insert Allowed`	Grant Insert Privilege on the table to the required role	Y/N Default Y
`Update Allowed`	Grant Update Privilege on the table to the required role	Y/N Default Y
`Delete Allowed`	Grant Delete Privilege on the table to the required role	Y/N Default Y
`Select Allowed`	Grant Select Privilege on the table to the required role	Y/N Default Y
`Truncate Allowed`	The value of this UDP indicates whether a TRUNCATE statement is allowed on the table	Y/N Default Y
`Maintain Partition`	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.	Y/N Default Y
`Exchange Partitions`	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.	Y/N Default Y
`Maintain Index`	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.	Y/N Default Y

Table 56–11 View Object Type Properties

UDP Name	Description	Values
`Insert Allowed`	Grant Insert Privilege on the view to the required role	Y/N Default Y
`Update Allowed`	Grant Update Privilege on the view to the required role	Y/N Default Y
`Delete Allowed`	Grant Delete Privilege on the view to the required role	Y/N Default Y
`Select Allowed`	Grant Select Privilege on the view to the required role	Y/N Default Y

Table 56–12 Sequence Object Type Properties

UDP Name	Description	Values
`Select Allowed`	Grant Select Privilege on the sequence to the required role	Y/N Default Y
`Reset Sequence`	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.	Y/N Default Y

Table 56–13 Materialized Views Object Type Properties

UDP Name	Description	Values
`Select Allowed`	Grant Select Privilege on the materialized view to the required role	Y/N Default Y

57 Improving Performance

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:

57.1 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 (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 Performance and Tuning Guide*.

57.2 ADF Business Components Guidelines

To maximize performance while working with ADF Business Components, such as entity objects, view objects, application modules, and services, consider best practices. For more information about tuning Oracle ADF, see the "Oracle Application Development Framework Performance Tuning" chapter in the *Oracle Fusion Middleware Performance and Tuning Guide*.

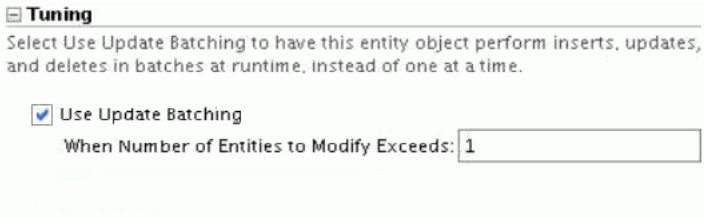
57.2.1 Working with Entity Objects

When working with entity objects, consider the following suggestions for improving performance. For more information see the "What You May Need to Know About Optimizing View Object Runtime" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

57.2.1.1 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 Figure 57–1. You should also set the **Threshold** property to *1*, which is important for `_TL` multi language entities.

Figure 57–1 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.

You should always enable batch updates, except in the following three cases where you should not:

- 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, see the "Batch Processing" section in the *Oracle Fusion Middleware Performance and Tuning Guide*.

57.2.1.2 Children Entity Objects in Composite Entity Associations Should not set 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.

57.2.1.3 Avoid Using List Validator Against Large Lists

When you use list validator, it scans the values in a linear fashion. Therefore, you should 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.

57.2.1.4 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 you should move the call to outside the loop.

57.2.1.5 Close Unused RowSets

There are various places in ADF Business Components that loop through all rowsets of a view object. You should 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`.

57.2.1.6 Use "Retain Association Accessor RowSet" when Appropriate

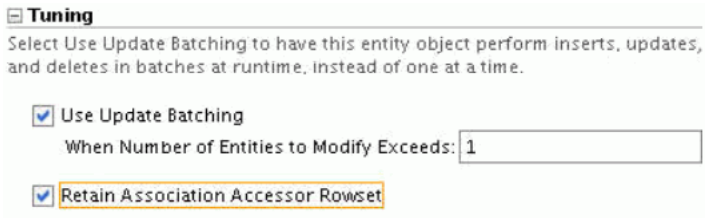
By default, the entity object creates a new `RowSet` object each time you retrieve an entity association accessor rowset, to allow you to work with the rows. However, creating a new `RowSet` object does not imply re-executing the query to produce the results each time, since only a new instance of a `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 hence the 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.

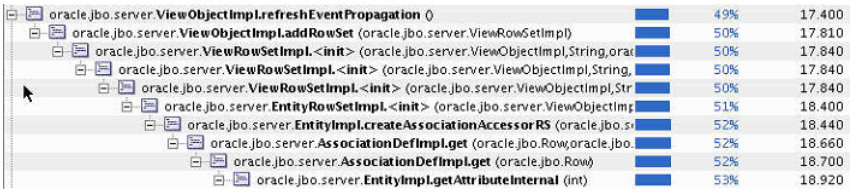
Using the **Retain Association Accessor RowSet** option may adversely affect memory usage since it postpones when the retained rowset becomes garbage collectible. Before you enable this option, (as shown in Figure 57–2), you should profile your flow to make sure you would indeed get a noticeable benefit.

Figure 57–2 Entity Object Editor — Tuning: Retain Association Accessor RowSet



If you see the top CPU consumers (sort by exclusive CPU) 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`. Figure 57–3 illustrates an example profiler output showing `addRowSet` being expensive.

Figure 57–3 Profiler Output Example



Before you decide to retain association accessor, you should try the guideline Section 57.2.1.5, "Close Unused RowSets."

If you decide to use the **Retain Association Accessor RowSet** option, you should be aware of the potential behavior changes. For more information, see the *Advanced Entity Association Techniques* section in the "Advanced Entity Object Techniques" chapter of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

57.2.1.7 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.

57.2.2 Working with View Objects

When working with view objects, consider the following suggestions for improving performance.

57.2.2.1 Tune the View Object SQL Statement

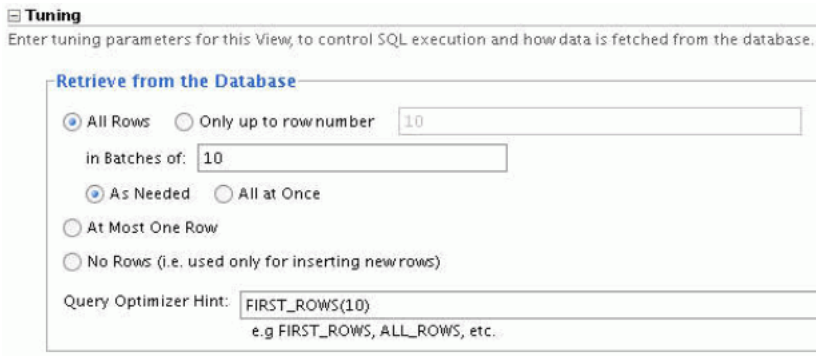
You should tune both the list of attributes included in the view object as well as the underlying SQL statement. Avoid using the "one-size fits all" view objects which include many other attributes that are not needed for your usage. These additional attributes consume unnecessary memory.

You should 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 should 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 Figure 57–4.

Figure 57–4 View Object Editor — Tuning



For user interface (UI) driven queries, the `FIRST_ROWS(10)` hint should be used to instruct the optimizer to pick a plan that is optimized to return the first set of rows. You should set this hint for view objects that are used for UI components, which typically just displays the initial set of rows (such as *table*). If you are fetching all the rows, then do not use the `FIRST_ROWS` hint.

57.2.2.2 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 *Oracle Fusion Middleware Performance and Tuning Guide*.

57.2.2.3 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. For more information about tuning the *fetch* options for the application, see the "Configuring View Object Data Fetching" section in the *Oracle Fusion Middleware Performance and Tuning Guide*.

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 the practice of applying a fixed fetch size, such as 30, to all view objects should be avoided.

If you have a view object that is used in both query and insert, then you should 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 pre-determined 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 should be 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. For more information, see the "Working with List of Values (LOV) in View Object Attributes" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

57.2.2.4 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 the "Additional View Object Configurations" section in the *Oracle Fusion Middleware Performance and Tuning Guide*.and the "Working with Bind Variables" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

57.2.2.5 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, in order to use a database index and avoid a full table scan. Otherwise, the SQL generated will be of the form:

```
((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.

57.2.2.6 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. For more information, see the "Configuring View Object Data Fetching" section in the *Oracle Fusion Middleware Performance and Tuning Guide*.

The `setForwardOnly` API is actually 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.

57.2.2.7 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, this call should be avoided. 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 1 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`.

57.2.2.8 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, you should 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, you should 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`.

57.2.2.9 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**.

57.2.2.10 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 Figure 57–5.

Figure 57–5 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 *Oracle Fusion Middleware Performance and Tuning Guide*.

57.2.2.11 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 does 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 Example 57–1.

Example 57–1 Use setListenToEntityEvents(false)

```
RowSet myRowSet = (RowSet) myEntityImpl.getAttribute("<Accessor Name>");
((ViewObjectImpl) MyRowSet.getViewObject()).setListenToEntityEvents(false);
```

57.2.2.12 Use Appropriate Getter or Setter on View Row

If you have a `ViewRowImpl` class generated for your view object, you should 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, you should consider using `getAttribute(int index)` and `setAttribute(int index)`.

57.2.2.13 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 as a result, expensive table scans can result if this view criteria item is supposed to be the driving filter. You should make sure there are appropriate function indexes to support case-insensitive searches.

57.2.2.14 Avoid View Object Leaks

In general, avoid creating a view object at runtime. You should 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, you should 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, you should 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.

57.2.2.15 Provide a "Smart" Filter when Using LOV Combobox

When you define a Combo Box with List of Values, you should 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.

57.2.2.16 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. You should not change the ListRangeSize to a large value. In particular, *-1* should never be used as it brings back all the rows.

57.2.2.17 Avoid Reference Entity Objects when not Needed

If your view object includes reference entity objects, they are loaded in 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, you should use a view object that do 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.

57.2.2.18 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.

57.2.2.19 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.

57.2.2.20 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.

57.2.3 Working with Application Modules

When working with application modules, consider the following suggestions for improving performance.

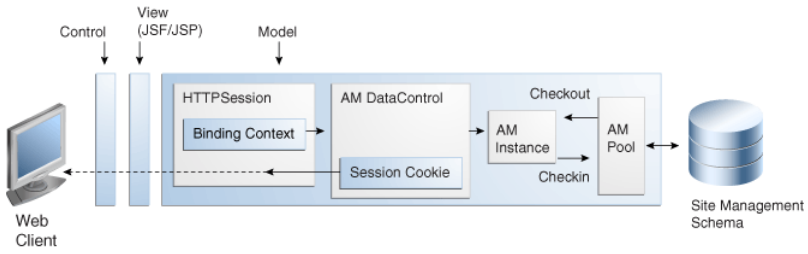
57.2.3.1 Enable Lazy Delivery

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 *Oracle Fusion Middleware Performance and Tuning Guide*.

57.2.3.2 Make Application Code Passivation-Safe

An application module is an ADF Business Components container that encapsulates business service methods and 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, a HTTP request, if related to data operation, can not be processed without involvement of an application module instance. Figure 57–6 illustrates the application module position in the Oracle ADF applications architecture.

Figure 57–6 Oracle ADF Applications Architecture — Application Module Functions



Application module state management and application module pooling are very 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 application 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.

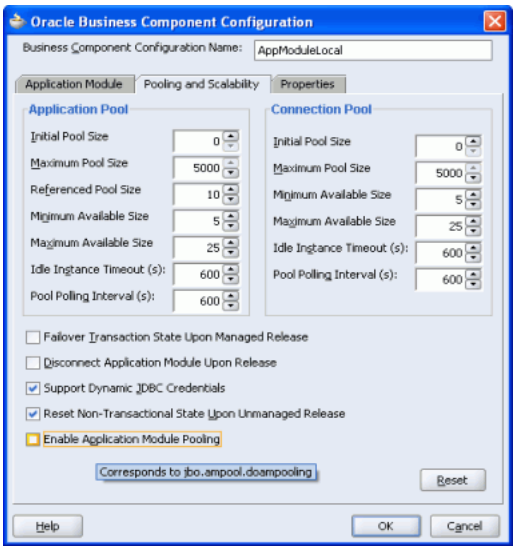
After coding and debugging all functional issues of an ADF Business Components based application, it is necessary to disable application module pooling to test and verify the application code is *passivation-safe*. Disabling application module pooling enforces the application module instance to be released 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. See Figure 57–7.
4. Deselect **Enable Application Module Pooling**.

Figure 57–7 Application Module: Configurations — Pooling and Scalability



For more information about application module state management, see the "Application State Management" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

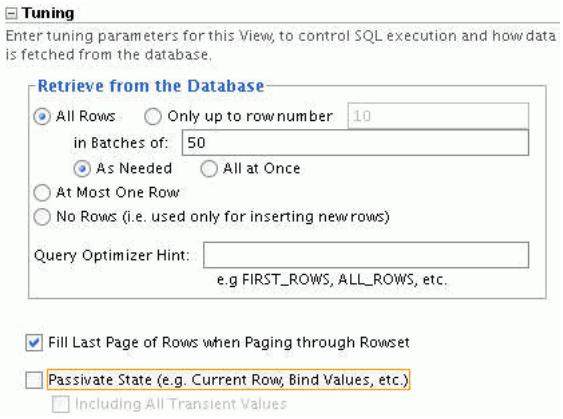
For more information about application module pooling, see the "Tuning Application Module Pools and Connection Pools" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

57.2.3.3 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 is no need to passivate LOV view objects and validator view objects where the bind values are coming from the target data row via the view accessor relationship. 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 you should disable passivation.

To disable passivation for a view object, uncheck the **Passivate State** option in the **View Object Editor**, as shown in Figure 57–8.

Figure 57–8 View Object Editor — Tuning



57.2.3.4 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 57–8. 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

57.2.3.5 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. On the other hand, 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 once the application module instance switching happens.

It is strongly suggested to 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.

For more information about how to manage custom user information, see the "Application State Management" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

The following is sample code from the Pathfinder Application to passivate and activate the `UserLoginPrincipal` object. Example 57–2 shows one way that you can passivate custom state.

> **Note:** XML documents can only handle `String`. Therefore, an object must be serialized before saving.

Example 57–2 Passivating and Activating UserLoginPrincipal

```
public void passivateState(Document doc, Element parent)
  {
      super.passivateState(doc,parent);

      UserLoginPrincipal principal = (UserLoginPrincipal)
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;
                       UserLoginPrincipal principal = null;
                            try
                            {
                              ois = new ObjectInputStream(bais);
                              principal = (UserLoginPrincipal)ois.readObject();
                            }
                            catch (IOException e)
                            {}
                            catch (ClassNotFoundException e)
                            {}
                            getSession().getUserData().put(Constants.LOGIN_
PRINCIPAL,principal);
                            break;
                          }
                  }
          }
      }
  }
```

57.2.3.6 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 `HTTPSession` 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: Once 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.

Table 57–1 illustrates application module release mode comparisons.

Table 57–1 Application Module Release Mode Comparison

Release Mode	Unmanaged (Stateless)	Managed (Stateful)	Reserved
Application Module Behavior	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. **Note:** 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).	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. **Note**: Stateful mode provides failover support for the HTTP session and is the preferred choice when the application module uses a standard JDBC connection.	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. **Note**: Reserved mode is primarily useful when the application module requires a non-standard JDBC connection definition. Failover is not supported in this mode.

Table 57–1 (Cont.) Application Module Release Mode Comparison

Release Mode	Unmanaged (Stateless)	Managed (Stateful)	Reserved
DBTransaction & User Action	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 **Commit** and **Rollback** buttons are disabled in the JSP page.	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 **Commit** button in the process JSP page. Once the user clicks the **Commit** button, Oracle ADF immediately initiates a post and commit (together, as one step) on the database. Optionally, the user can click the **Rollback** 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.	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 **Commit** button or **Rollback** 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.
Application Module Locking Behavior	In stateless mode, it is recommended that the Business Components property `jbo.locking.mode` should be set to *optimistic*. 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.	In stateful mode, it is recommended that the Business Components property `jbo.locking.mode` should be set to *optimistic*. 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.	In release mode, you can reliably use pessimistic locking and may set the property `jbo.locking.mode` to *pessimistic*.

For more information about application module release level and state management, see the "Application State Management" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

57.2.3.7 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, 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.

57.2.3.8 Release Dynamically Created Root Application Modules

If you create an application module using createRootApplicationModule calls, you should 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 *AMImpl.getInstance calls for the various Applcore application modules.

57.2.3.9 Do Not Destroy the Application Module when Calling Configuration.releaseRoot ApplicationModule.

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 needs to 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.

57.2.4 Working with Services

When working with services, consider the following suggestions for improving performance.

57.2.4.1 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, then make sure you set the find criteria to fetch only the attributes you need.

Example 57–3 is sample code showing how to create a find criteria.

Example 57–3 How to Create Find Criteria

```
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.INSTANVCE.create(ViewCriteraItem.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
```

```
  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);
  fc.setChildFindCriteria(cfcl);
  DeptResult dres = svc.fndDept(fc, null);
  pw.println("### Dept 10 and 2nd Emp ###");
.......
```

57.2.4.2 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.

57.2.4.3 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 use `ChangeOperation.MERGE`. Calling the `processXXX()` method with `ChangeOperation.MERGE` issues extra queries to the database to check if the rows already exist.

57.2.4.4 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, if possible, set only the columns that you really need to change. Service data objects with fewer attributes that have been set are updated faster than service data objects with all the attributes set.

57.3 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.

57.3.1 Working with Various ADF ViewController Components

When working with ADF ViewController components, consider the following suggestions for improving performance.

57.3.1.1 Minimize the Number of Application Module Data Controls

For a specific page or page fragment, try to use only one application module data control. You should use nested application modules rather than a separate application module data control because this minimizes 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.

57.3.1.2 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 *Oracle Fusion Middleware Performance and Tuning Guide*.

57.3.1.3 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.

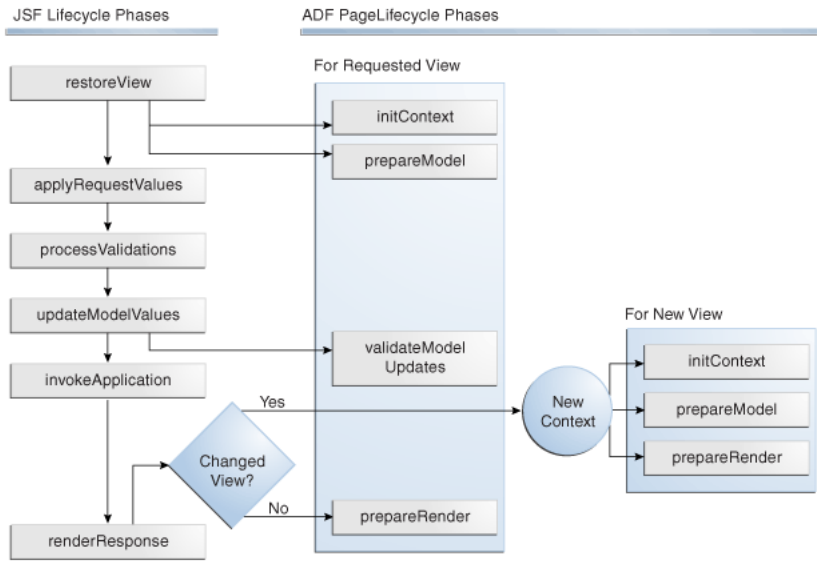
57.3.1.4 Disable Column Stretching

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.

57.3.1.5 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 value *ifNeeded* for the `Refresh` property, the iterator or action may get refreshed twice and therefore impact performance. Figure 57–9 shows how the JavaServer Faces (JSF) and Oracle ADF phases integrate in the lifecycle of a page request.

Figure 57–9 Lifecycle of a Page Request in an Oracle Fusion Web Application



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 `invokeAaction` binds to the `methodAction`, which does not accept any parameters,

or to any action then it will fire 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

57.3.1.6 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. This turns off the row count query.

You 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.

57.3.1.7 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 `HTTPSession` cache is not recommended. Instead, you should leverage application module state management and application module pooling. See Section 57.2.3.2, "Make Application Code Passivation-Safe."

Example 57–4 shows how to use `HTTPSession` cache in a backing bean.

Example 57–4 HTTPSession Cache in a Backing Bean

```
((HttpServletRequest)
  (FacesContext.getCurrentInstance().getExternalContext().getRequest())))
  .getSession().setAttribute("UserLoginPrincipal",sessionLoginPrincipal);
```

57.3.1.8 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.

57.3.1.9 Follow UI Standards when Using Search

When using Search, follow these UI standards:

- Blind queries are not allowed.
- **Match All** should be used instead of **Match Any** when there are multiple criteria.

57.3.1.10 Avoid Executing Component Subtree by Adding a Condition Check

In some cases it is possible to find out during the jsp tag execution phase if a particular jsp subtree needs to be executed or not by using the <c:if test...> tag. Example 57–5 is an example for `panelAccordion`. (Note the use of `$` instead of `#`).

> **Note:** Using this technique is not recommended. For other techniques, see Section 57.3.1.20, Section 57.3.1.21, and Section 57.3.1.22.

Example 57–5 Using the <c:if test...> Tag

```
<af:panelAccordion ......>
  <af:showDetailItem disclosed="#{item.disclosed}" .......>
    <c:if test="${item.disclosed}">
      <!--Content here will not be executed in jsp engine if item is not
disclosed-->
    </c:if>
  </af:showDetailItem>
</af:panelAccordion>
```

Example 57–6 shows how to use this technique with lazy popups.

Example 57–6 Using <c:if test...> Tag with Lazy Popups

```
<af:popup id="popupRegion" contentDelivery="lazyUncached"
                  launcherVar="source" eventContext="launcher">
       <!-- param passed to taskflow -->
       <af:setPropertyListener from="#{source.attributes.param}"
                               to="#{requestScope.param}" type="popupFetch"/>
       <!-- reset taskflow and turn on activateSubtree (session scoped) -->
       <af:setPropertyListener from="Y" to="#{testBean.refreshTaskflow}"
                               type="popupFetch"/>
       <af:panelWindow id="window" title="Task Flow">
            <!-- conditionally includes the nested components -->
            <c:if test="${testBean.activateSubtree}">
              <af:region value="#{bindings.dynamicRegion1.regionModel}"
                         id="dynamicRegion1"

regionNavigationListener="#{testBean.navigationListener}"/>
            </c:if>
       </af:panelWindow>

        <!-- toggles off the activateSubtree flag -->
        <af:serverListener type="serverPopupClosed"
                           method="#{testBean.popupClosedListener}"/>
        <!-- queues a custom event on close of the popup to invoke the
```

```
serverListener -->
        <af:clientListener method="popupClosedListener" type="popupClosed"/>
</af:popup>
```

57.3.1.11 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.

57.3.1.12 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 *Oracle Fusion Middleware Performance and Tuning Guide*.

57.3.1.13 Use Appropriate ContentDelivery 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 *Oracle Fusion Middleware Performance and Tuning Guide*

57.3.1.14 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, you should set iterator range size to the value of autoHeightRows + 1.

57.3.1.15 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, make sure the row height of the columns to the left and the right of the frozen column are of the same height.

57.3.1.16 Avoid Unnecessary Regions

Regions are very powerful and provide extreme flexibility. However, there is an associated cost with every region. In order to have the best performance, make sure to use the region only when it is needed.

Generally, the Oracle ADF guideline is to not have more the 15 regions on the page.

57.3.1.17 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 connection. There may be some cases where using *Isolated* is functionally necessary. For more information, see the "Sharing Data Control Instances" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

57.3.1.18 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.

57.3.1.19 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.

57.3.1.20 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 as long as 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 these 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.

57.3.1.21 Delay Creation of Popup Child Components

This recommendation is similar to Section 57.3.1.20, "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 these 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.

57.3.1.22 Avoid Unnecessary Task Flow Activation for Regions Under Switchers

By default, task flows that use an `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:

```
<af:switcher id="s1" defaultFacet="1" facetName="#{pageFlowScope.facet}">
<f:facet name="1">
<af:region value="#{bindings.TF1.regionModel}" id="r1"/>
</f:facet>
<f:facet name="2">
<af:region value="#{bindings.TF2.regionModel}" id="r2"/>
</f:facet>
</af:switcher>
```

The associated binding should have activation set to "conditional", and active set to an EL, as follows:

```
<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"/>
```

57.3.1.23 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 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 Configuration.createRootApplicationModule() API without calling releaseRootApplicationModule().

You should 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. You should 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:

```
private ComponentReference<RichTable> allocationTableRef;
public void setAllocationTable(RichTable allocationTable) {
if( this.allocationTableRef == null)
this.allocationTableRef =
ComponentReference.newUIComponentReference(allocationTable);}
public RichTable getAllocationTable() {
return allocationTableRef==null ? null : allocationTableRef.getComponent();
```

The following example is not recommended:

```
private RichTable allocationTable;
public void setAllocationTable(RichTable allocationTable) {
this.allocationTable = allocationTable; }
public RichTable getAllocationTable() {
return allocationTable; }
```

57.3.1.24 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. You should check the box to avoid the savepoint cost if you do not have functionality to rollback to this particular savepoint.

57.3.1.25 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 should be 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.

57.3.1.26 Do Not Maintain References to UI Components in Managed Beans

If you maintain direct references to UI Component instances from view scope or pageflow scope beans, this could cause both functional errors and impact performance. If you must maintain a reference, use the ComponentReference pattern instead.

57.3.2 Enable ADF Rich Client Geometry Management

ADF Rich Client supports Geometry Management of the browser layout where parent components in the UI explicitly size the children components to stretch and fill up available space in the browser. While this feature makes the UI look better, it has a cost. For more information, see the "Enable ADF rich client geometry management" row in the "Configuration Parameters for ADF Faces" table in the *Oracle Fusion Middleware Performance and Tuning Guide*.

57.3.3 Use Page Templates

Page templates allow you to build reusable, data-bound templates that can be used as a shell for any page. For important considerations when using templates, see the "Use page templates" row in the "Configuration Parameters for ADF Faces" table in the *Oracle Fusion Middleware Performance and Tuning Guide*.

57.3.4 Use ADF Rich Client Partial Page Rendering (PPR)

You should always consider using partial page refresh instead of a full page refresh. For more information, see the "Use ADF Rich Client Partial Page Rendering" row in the "Configuration Parameters for ADF Faces" table in the *Oracle Fusion Middleware Performance and Tuning Guide*.

57.4 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 *Oracle Fusion Middleware Performance and Tuning Guide*.

57.5 Oracle Fusion Middleware Extensions for Applications Guidelines

When working with application modules, consider these best practices related to using a nested service and releasing application modules returned from `getInstance` calls.

57.5.1 Use Profile.get to Get Profile Option Values

To get a profile option value, use

```
oracle.apps.fnd.applcore.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.

57.5.2 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 Example 57–7.

Example 57–7 Release Application Module

```
ProfileServiceAM profileService = null;

try {
  profileService = ProfileServiceAMImpl.getInstance();
  String value = profileService.getProfileValue("<ProfileOptionName>");
}
finally {
  Configuration.releaseRootApplicationModule(profileService, false);
}
```

57.5.3 Avoid Unnecessary Activation of Attachments Taskflow

When the attachments feature is used, it creates a new taskflow in the page bindings. For example:

```
<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}"

57.5.4 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.

57.5.5 Set the Data Control Scope to Isolated for Page Level Item Nodes

If the data control scope is shared for taskflows pointed to by certain item nodes, then the life span of these taskflow 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 no removed for the duration of the session, regardless of any navigation or closing tab. This is due to the fact that Main TF and Regional TF in the UI shell has 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 taskflows 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 taskflow itself.

57.6 General Java Guidelines

When working with Java, consider these best practices related to Strings and StringBuilder, Collections, Synchronization, as well as other Java features.

57.6.1 Working with Strings and StringBuilder

When working with `Strings` and `StringBuilder`, consider the following suggestions for improving performance.

57.6.1.1 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 a `toString()`.

This increases cost in both space and time, especially if you're appending more than one `String`. You should 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:

```
String key = granteeType + ":" + granteeKey;
```

You should follow this example:

```
String key = new StringBuilder(granteeType.length() + 1 +
  granteeKey.length()).append(granteeType).append(":").append(granteeKey)
  .toString();
```

This way, 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, as long as you follow the logging standard and check 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:

```
if (formattedNumericValue.substring(0,1).equals("-")) negValue = true;
```

Do this instead:

```
if (formattedNumericValue.charAt(0) == '-') negValue = true;
```

The `hashCode` method is another common place where you do `String` concatenation. Example 57–8 uses the `hashCode` implementation and requires String concatenation on every call.

Example 57–8 Hashcode with String Concatenation

```
public int hashCode()
{
  … … …
    h = new StringBuffer(len).append(resp).append(rapl).toString().hashCode();
    return h;
}
```

Example 57–9 does not use String concatenation.

Example 57–9 Hashcode without 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.

57.6.1.2 Check the Log Level Before Making a Logging Call

You should always check the log level before you 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. For more information see the "Set Logging Levels" section in the *Oracle Fusion Middleware Performance and Tuning Guide.*

57.6.1.3 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.

57.6.1.4 Lazy Instantiation

Avoid instantiating objects until they are needed. 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 needs to be replaced. For more information, see the "Application Module Design Considerations" section in the *Oracle Fusion Middleware Performance and Tuning Guide*.

57.6.2 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 *Oracle Fusion Middleware Performance and Tuning Guide*.

57.6.3 Manage Synchronization

When working with synchronization methods you should 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.

57.6.4 Work with Other Java Features

When working with Java features, you should consider the following:

57.6.4.1 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 myInterger = 1000;
```

Example 57–10 shows how the compiled code basically creates a new Integer object based on the int value:

Example 57–10 Compiled Code

```
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.

57.6.4.2 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`.

57.6.4.3 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 needs to be created each time.

57.6.4.4 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:

```
if (methodA() >= 0)
  return methodA() + methodB();
```

Use:

```
int res = methodA();
If (res >= 0)
  return res + methodB();
```

57.6.4.5 Close Unused JDBC Statements to Avoid Memory Leaks

To avoid memory leaks, closed unused JDBC statements. Example 57–11 depicts a statement leak in java code.

Example 57–11 Statement Leak in Java Code

```
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 Example 57–12.

Example 57–12 Underlying Statement Not Closed

```
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, you should close it in the final block of a `try-catch`, as shown in Example 57–13.

Example 57–13 Close Statement Example

```
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()`.

57.6.4.6 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)
```

57.6.4.7 Avoid JDBC Connection Leaks

If you are getting a connection directly from Data Source or through `ApplSessionUtil.getConnection`, make sure you release the connection in a final block.

57.7 Caching Data

Caching is one of the most common approaches for improving the performance of accessing commonly used data. Shared application module and view object 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.

57.7.1 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.

57.7.2 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`.

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.

   For information about how to create a shared application module, see the "Sharing Application Module View Instances" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
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 Example 57–14.

Example 57–14 Pre-load all Data into Memory

```
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.

57.7.3 How to Cache Multi-Language Support Data

In addition to the instructions provided in Section 57.7.2, "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.

57.7.3.1 Creating 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.

57.7.3.1.1 How to create objects if only the data from the base table needs to 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.

57.7.3.1.2 How to create objects if only the data from the _TL table needs to be shared

1. Create the entity object on top of the `_TL` table.

   This is required by MLS Framework. For more information, see Section 9.2, "Using 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.

57.7.3.1.3 How to create objects if both the data from the base table and the _TL table needs to be shared

1. Create the entity object on top of the `_TL` table.

This is required by MLS Framework. For more information, see Section 9.2, "Using 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.

57.7.3.2 Creating 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.

57.7.3.2.1 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.

57.7.3.2.2 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.

57.7.3.2.3 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.

57.7.4 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 your use case requires it.

57.7.4.1 Consuming 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.

   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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
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 utilize 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.

57.7.4.2 Creating 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:

```
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 in order to get the latest data. For more information, see the "Sharing Application Module View Instances" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

57.7.5 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

57.8 Profiling and Tracing 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 Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework.

57.8.1 How to Profile Oracle Fusion Applications with JDeveloper Profiler

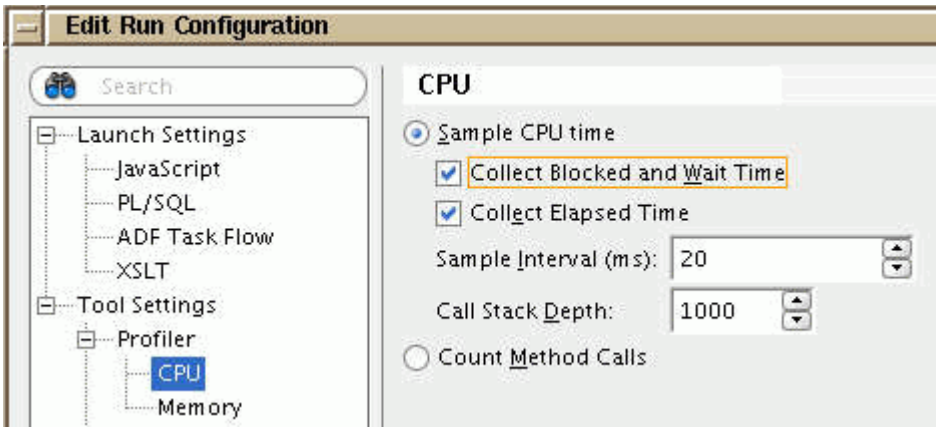
The JDeveloper Profiler is used to provide information about the CPU, elapsed time, and memory metrics, as well as call counts. It can be very helpful when you are dealing with a performance issue or just trying to understand the performance characteristics of your code.

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.

Figure 57–10 Edit Run Configuration — Profiler: CPU



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, you should set appropriate filters to include only the classes you are interested in.

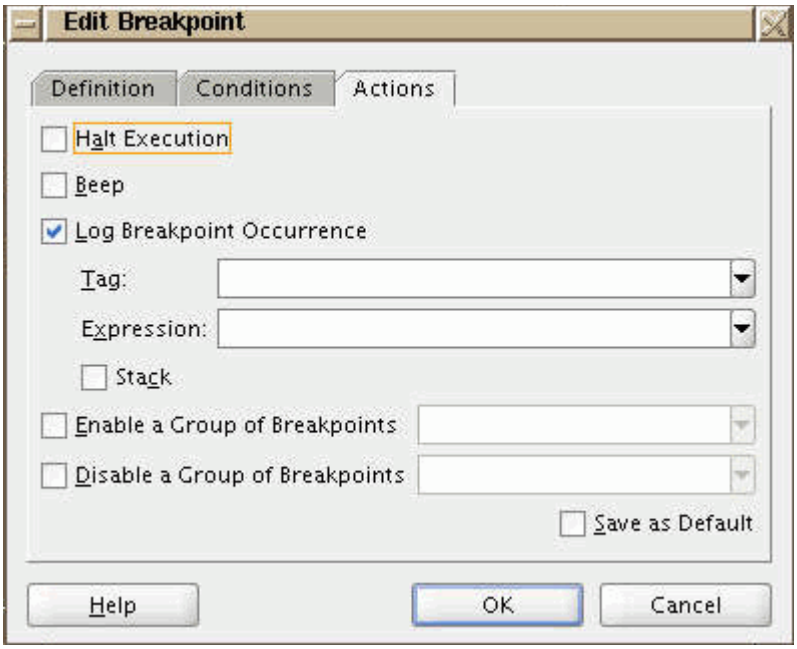
57.9 Set up a Debug Breakpoint

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 Figure 57–11.

Figure 57–11 Edit Breakpoint

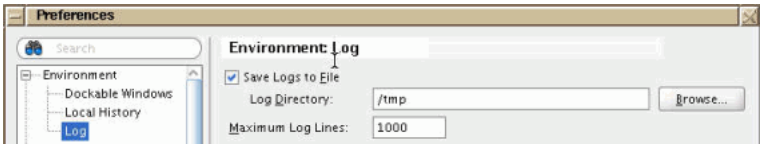


Each time the breakpoint is hit, the stack is written to the console. To capture this, you must log the console output to a file.

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 Figure 57–12.

Figure 57–12 Preferences — Environment: Log



After running your project, you can find the console logged to a file in the specified directory.

58

Debugging Oracle ADF and Oracle SOA Suite

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:

58.1 Introduction to Debugging Oracle ADF Debugging 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.

58.2 Collecting Diagnostics

Collecting diagnostics information helps you to obtain more contextual information for isolating the problem.

58.2.1 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

58.2.1.1 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`. For information, see the "How to Turn On Diagnostic Logging" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

58.2.1.2 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 Relational Database Management System (RDBMS). 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:

  ```
  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 RDBMS 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 RDBMS Support and Development diagnose the cause. Both the alert log and the trace file will be in the RDBMS User Directory.

You can also enable Oracle ADF tracing. For information, see the "Use SQL Tracing to Identify Ill-Performing Queries" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

58.2.2 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

58.2.2.1 Enabling 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 Chapter 54, "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`:

```
<logger name='oracle.apps' level='ALL'
useParentHandlers='false'>
  <handler name='apps-handler'/>
</logger>
```

   The handler `apps-handler` should also exist in the handlers section.

58.2.2.2 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.

58.2.2.3 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.

58.2.2.3.1 Enabling 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.

58.2.2.3.2 Locating 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`*`/`*`sid`*`/trace`. The trace filename includes the FND session ID appended to the end, for example, `mysid_ora_4473_881497BF7770BEEEE040E40A0D807BB1.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..:1249055856737`.

   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.

58.3 Diagnosing Problems

In addition to reviewing the diagnostics information, you can perform various tasks to diagnose problems in your server environment.

58.3.1 How to Diagnose Problems in the Integrated WebLogic Server Environment

Perform the following tasks to diagnose problems in the Integrated WebLogic Server environment:

- Test the JDBC data source connections

- View the application module pooling statistics
- Sanity check your enterprise archive (EAR) file

58.3.1.1 Testing 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.

58.3.1.2 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. For more information about application module pools, see the "Tuning Application Module Pools and Connection Pools" chapter in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

To view the application module pooling statistics:

1. Point your browser to `http://localhost:7101/dms/Spy`.
2. Log in as an administrator.
3. Click the `ampool` metric.
4. Scroll your browser to the right to view all the statistics.

58.3.1.3 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.

58.3.2 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

58.3.2.1 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 Section 58.3.1.3, "Sanity Checking Your EAR File in the Integrated WebLogic Server Environment".

58.3.2.2 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.

58.4 Debugging in JDeveloper

The following debugging tools are available for debugging in JDeveloper:

- JDeveloper Debugger.

  For information, see the JDeveloper online help.

- ADF Declarative Debugger.

  For information, see the "Using the ADF Declarative Debugger" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

58.4.1 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/system11.1.1.1.32.52.37/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:

      JDEV_system_dir/SvcValidation/Supplier/src/oracle/apps/sv/supplier/supplierService/applicationModule/server/SupplierServiceImpl.java

   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.

58.5 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.

58.5.1 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
- Oracle WebLogic Server Hot Reloading Does Not Work
- Missing ADF Component at Runtime in Oracle WebLogic Server
- Odd ADF Component Errors
- Oracle WebLogic Server 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 Oracle Fusion Applications Database
- "The task cannot be processed further" Message Appears

- TimedOut Exception Occurs

58.5.1.1 "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 Section 2.2.1.2, "Increasing Open File Limit on Local Linux Servers".

58.5.1.2 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.

> **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.

58.5.1.3 "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.

58.5.1.4 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.

58.5.1.5 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.

58.5.1.6 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.

58.5.1.7 "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.

58.5.1.8 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=ku8guslcz_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.

58.5.1.9 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.

58.5.1.10 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.

58.5.1.11 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.

58.5.1.12 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.

58.5.1.13 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.

58.5.1.14 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.

58.5.1.15 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.

58.5.1.16 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`:

```
<context-param>
    <description>
        This parameter turns on the session change persistence.
    </description>
    <param-name>org.apache.myfaces.trinidad.CHANGE_PERSISTENCE</param-name>

 <param-value>oracle.adf.view.rich.change.MDSDocumentChangeManager</param-value>
</context-param>
```

Change the value of `CHANGE_PERSISTENCE` to `oracle.adf.view.page.editor.change.ComposerChangeManager`.

58.5.1.17 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."

58.5.1.18 "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.

58.5.1.19 TimedOut Exception Occurs

You get the following exception:

```
weblogic.transaction.internal.TimedOutException
```

Problem

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.

1. Launch the Administration Server instance of Oracle WebLogic Server (`http://localhost:7101/console`).
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`.

58.6 Testing and Troubleshooting Oracle SOA Suite

For more information about testing and troubleshooting SOA composite applications, see the "Testing and Troubleshooting" section in *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

You can also automate the SOA composite applications testing. For information, see the "Automating Testing of SOA Composite Applications" section in *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

59

Designing and Securing View Objects for Oracle Business Intelligence Applications

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:

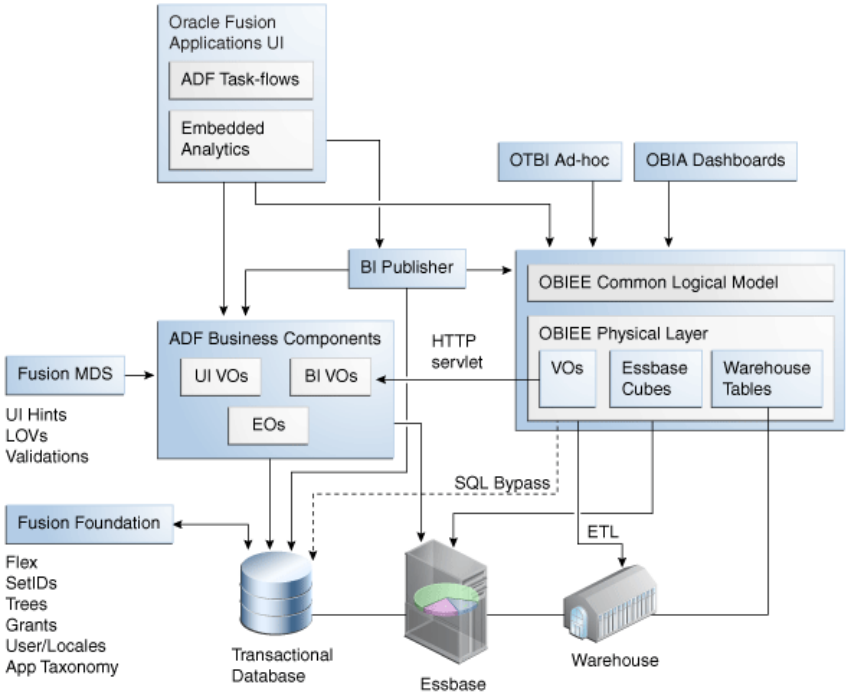
59.1 Introduction to View Objects for Oracle Business Intelligence Applications

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 Fusion application 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 Applications tables. The extract from the source system is done using the Oracle Application Development Framework (Oracle ADF) view objects.

Figure 59–1 illustrates the Oracle Business Intelligence architecture.

Figure 59–1 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 in order 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, in order 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.

59.2 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.

59.2.1 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

All attributes from the physical table (with the exception of special, highly sensitive attributes) should be exposed on the entity objects.

59.2.2 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. For more information about flattening, see Section 59.3.1, "Understanding Flattened View Objects."

59.2.3 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 *Oracle Fusion Middleware Fusion Developer's Guide for 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.

59.2.3.1 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 *Oracle Fusion Middleware 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 *Oracle Fusion Middleware Fusion Developer's Guide for 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.

59.2.3.2 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 59–1 shows the Standard Who Columns.

Table 59–1 Standard Who Columns

Standard Who Columns	Description
`CREATED_BY`	The user who created the row.
`CREATION_DATE`	The date and time the row was created.
`LAST_UPDATED_BY`	The user who last updated the row.
`LAST_UPDATE_DATE`	The date and time the row was last updated.

Table 59–1 (Cont.) Standard Who Columns

Standard Who Columns	Description
LAST_UPDATE_LOGIN	The session login associated to the user who last updated the row.

- 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 should resolve the foreign key using entity object associations. For MLS-enabled entities, ID and Code attributes should be resolved using _VL views.

59.2.3.3 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 Section 59.4.1, "Designing Fact View Objects.") If a join is required to resolve an ID or Code attribute, use a _VL view instead.

59.2.4 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

59.2.5 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 Section 59.3.4, "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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

59.3 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.

59.3.1 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.

59.3.2 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.

59.3.3 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 `CreditdInvoicesVO`, 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.

59.3.4 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 an 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.

59.3.5 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.

59.3.6 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

59.3.6.1 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.

59.4 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.

59.4.1 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 59–2.

Table 59–2 Flattened View Objects Based on Fact Grains Example

Flattened View Objects	Fact Grains
POHeaderVO	Header
POLinesVO	Header + Lines
POShipmentsVO	Header + Lines + Shipments
PODistributionsVO	header + Lines + Shipments + Distributions

Entity objects can be included in flattened view objects as required, as long as 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.

59.4.2 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 Section 48.3.2, "How to Secure Rows Queried By Entity-Based View Objects."

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"
```

59.4.2.1 Securing 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:

"for the participants for who they are responsible" OR "for business units for which they are authorized"

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 59–1 assuming IC_INCENTIVE_COMPENSATION_PAYSHEET is the object registered in FND_OBJECT and ICPAY is the alias used for the entity object.

Example 59–1 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 end 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.

59.4.2.2 Securing 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.

59.4.2.3 Joining 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. As a result, 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; where as the desired result was just to apply security of Fact B.

59.4.2.4 Securing 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.

59.5 Designing and Securing Dimension View Objects

This section discusses how to design and secure dimensions.

59.5.1 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.

59.5.2 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.

59.5.3 Securing Dimension View Objects

If the dimension needs to be secured, then the FND view criteria should be applied on the dimension view object.

59.5.3.1 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* should be used. In other words, data security on HR entities should be ignored when consumed in `InvOrg`.

This use case is similar to Section 59.4.2.2, "Securing Transactions Different from Securing Dimensions," where unsecured view objects are used for dimensions.

59.5.3.2 Securing Transactions Using Dimension with Dimension Browsing Unsecured

The Dimension Business Unit is used to secure transaction data. When used in conjuction 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.

59.5.4 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.

59.5.5 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.

59.5.6 Using Secured and Unsecured Dimension View Objects

There are a number of 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.

59.6 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.

59.6.1 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.

59.6.2 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.

59.6.3 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.

59.6.4 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.

59.6.5 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.

59.7 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.

59.7.1 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.

59.8 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:

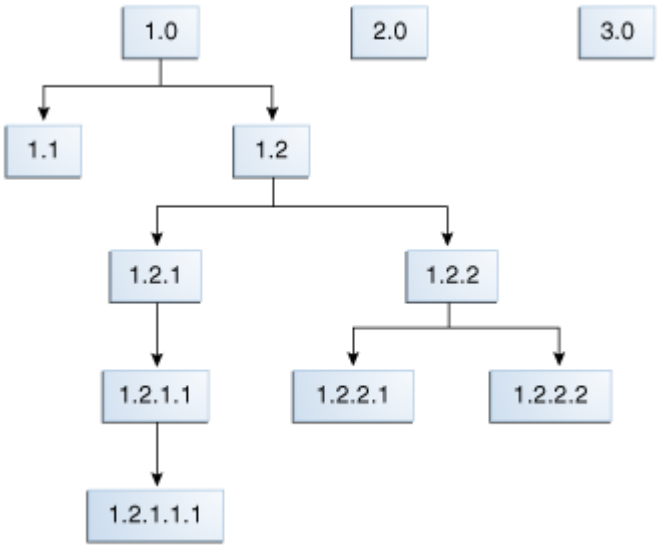
- Designing a column-flattened view object
- Customizing the FND table structure and indexes
- Using declarative SQL mode to design view objects
- Securing Oracle ADF view objects for trees

59.8.1 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 59–2 illustrates a generic example of a value-based tree.

Figure 59–2 Value-Based Tree Example



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 59–2, 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 59–3.

Note: In practice, you would never have single node trees. However, root nodes 2.0 and 3.0 are in Figure 59–2 to simply illustrate multiple top-nodes.

Table 59–3 Column-Flattened Representation of the Value-Based Tree Example

C0	C1	C2	C3	C4	Distance
1.0	1.0	1.0	1.0	1.0	0
1.1	1.1	1.1	1.1	1.0	1
1.2	1.2	1.2	1.2	1.0	1
1.2.1	1.2.1	1.2.1	1.2	1.0	2
1.2.1.1	1.2.1.1	1.2.1	1.2	1.0	3

Table 59–3 (Cont.) Column-Flattened Representation of the Value-Based Tree Example

C0	C1	C2	C3	C4	Distance
1.2.1.1.1	1.2.1.1	1.2.1	1.2	1.0	4
1.2.2	1.2.2	1.2.2	1.2	1.0	2
1.2.2.1	1.2.2.1	1.2.2	1.2	1.0	3
1.2.2.2	1.2.2.2	1.2.2	1.2	1.0	3
2.0	2.0	2.0	2.0	2.0	0
3.0	3.0	3.0	3.0	3.0	0

The following conventions apply to the logical column-flattened representation shown in Table 59–3.

- 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 a number of 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). On the other hand, 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 59–4.

Table 59–4 Column-Flattened Value-Based Tree Fixed at 20 Levels

C0	...	C15	C16	C17	C18	C19	Distance
1.0	...	1.0	1.0	1.0	1.0	1.0	0
1.1	...	1.1	1.1	1.1	1.1	1.0	1

Table 59–4 (Cont.) Column-Flattened Value-Based Tree Fixed at 20 Levels

C0	...	C15	C16	C17	C18	C19	Distance
1.2	...	1.2	1.2	1.2	1.2	1.0	1
1.2.1	...	1.2.1	1.2.1	1.2.1	1.2	1.0	2
1.2.1.1	...	1.2.1.1	1.2.1.1	1.2.1	1.2	1.0	3
1.2.1.1.1	...	1.2.1.1.1	1.2.1.1	1.2.1	1.2	1.0	4
1.2.2	...	1.2.2	1.2.2	1.2.2	1.2	1.0	2
1.2.2.1	...	1.2.2.1	1.2.2.1	1.2.2	1.2	1.0	3
1.2.2.2	...	1.2.2.2	1.2.2.2	1.2.2	1.2	1.0	3
2.0	...	2.0	2.0	2.0	2.0	2.0	0
3.0	...	3.0	3.0	3.0	3.0	3.0	0

Think of the tree in Figure 59–2 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 59–5, assuming the same "pad toward leaf values" scheme as with the value-based tree.

Table 59–5 Column-Flattened Level-Based Tree Rooted at Node 1.0

C0	C1	C2	C3	C4	Distance
1.1	1.1	1.1	1.1	1.0	1
1.2.1.1.1	1.2.1.1	1.2.1	1.2	1.0	4
1.2.2.1	1.2.2.1	1.2.2	1.2	1.0	3
1.2.2.2	1.2.2.2	1.2.2	1.2	1.0	3

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).

59.8.1.1 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 **Dep*N***, 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 Section 59.8.4, "Guidelines for ATG-Registration and BICVO Generation" and Section 59.8.6, "Securing ADF Business Components View Objects for Trees."

To generate BICVO automatically using Tree Management:

1. Ensure that the namespace path `/oracle/apps/fnd/applcore/trees/analytics` is configured in Oracle Metadata Service (MDS). Example 59–2 shows a sample MDS configuration.

Example 59–2 MDS Configuration

```
<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"/>
         <namespace path="/oracle/apps/fnd/applcore/trees/analytics"
             metadata-store-usage="mdsRepos"/>
      </metadata-namespaces>
      <metadata-store-usages>
         <metadata-store-usage id="mdsRepos">
           <metadata-store name="fs1" class-name="oracle.mds.persistence.
             stores.file.FileMetadataStore">
           <property name="metadata-path" value="/tmp"/>
           </metadata-store>
          </metadata-store-usage>
         </metadata-store-usages>
     </persistence-config>
  </mds-config>
</adf-mds-config>
```

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 Section 19.3.5, "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 Section 59.8.6, "Securing ADF Business Components View Objects for Trees."

   The generated BICVO includes a special view criteria named `FNDDS__BICVO`. In order 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.

59.8.2 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 exactly 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.

59.8.3 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:

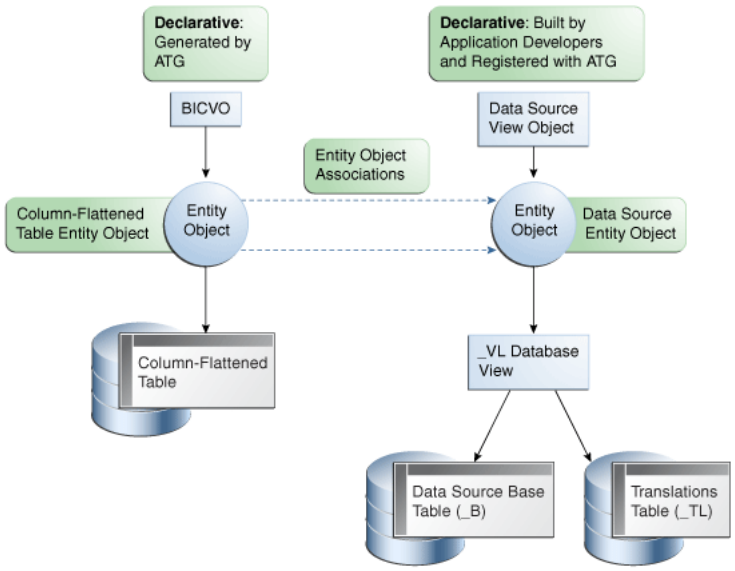
- Design Pattern #1: Single data source view object, single data source entity object
- Design Pattern #2: Multiple data source view objects, unique data source view object per depth of tree, single data source entity object per data source view object

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.

59.8.3.1 Using Single Data Source View Object Design Pattern

This pattern ensures that Oracle Fusion tree management is capable of generating a declarative-mode BICVO from an Oracle Applications Technology (ATG)-registered data source. Figure 59–3 illustrates the ADF Business Components object configuration defining declarative BICVO pattern #1.

Figure 59–3 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.

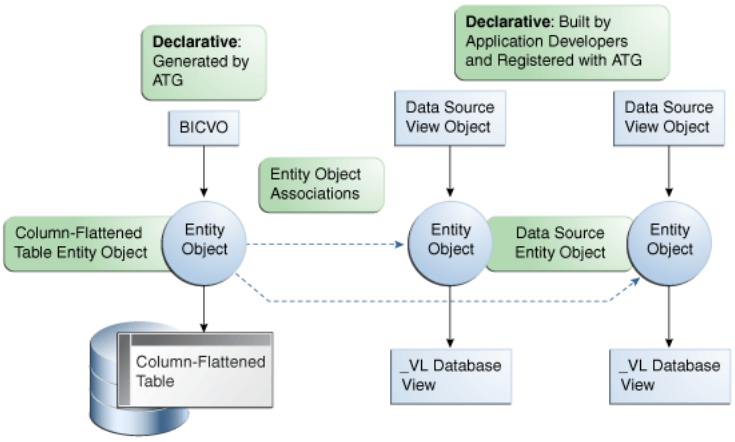
Once the data source view object is registered with Oracle Fusion tree management as part of the tree structure definition process, and the required manually-created objects are all in place, a declarative BICVO may then be generated by Oracle Fusion tree management.

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.

59.8.3.2 Using Multiple Data Source View Objects Design Pattern

This pattern ensures that Oracle Fusion tree management is capable of generating a declarative-mode BICVO from an ATG-registered tree. Figure 59–4 illustrates the ADF Business Components object configuration defining declarative BICVO Pattern #2.

Figure 59–4 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.

59.8.3.3 Setting the Declarative-Mode BICVO Properties

In order 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*.

59.8.4 Guidelines for ATG-Registration and BICVO Generation

The Oracle Fusion Applications team 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 Applications team 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 59–3. 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 Applications team must then create both data source view objects and associated data source entity objects, according to either of the structural patterns illustrated in Figure 59–3 and Figure 59–4. 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. As a result, 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 59–3 and Figure 59–4, 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.applcore.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. As long as the interfaces of these objects are publicly visible, this should not present any problems to clients of these objects.

59.8.5 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 59–6 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 59–6 Mapping Oracle Business Intelligence Hierarchy Levels to Application Tree Levels

BI (RPD)	Application (BICVO)
Top	C31
Level Top +1	C30
.....	.....
Level Base - 1	C18
Base	C0

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 Intelligence side 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.

59.8.5.1 Resolving 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 Intelligence side (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.

59.8.6 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 59–3) 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 59–7 in bold.

Table 59–7 Column-Flattened Result Set with Data Security

C0	C1	C2	C3	C4	Distance
1.0	1.0	1.0	1.0	1.0	0
1.1	**1.1**	**1.1**	**1.1**	**1.0**	**1**
1.2	1.2	1.2	1.2	1.0	1
1.2.1	1.2.1	1.2.1	1.2	1.0	2
1.2.1.1	1.2.1.1	1.2.1	1.2	1.0	3
1.2.1.1.1	1.2.1.1	1.2.1	1.2	1.0	4
1.2.2	**1.2.2**	**1.2.2**	**1.2**	**1.0**	**2**
1.2.2.1	1.2.2.1	1.2.2	1.2	1.0	3
1.2.2.2	1.2.2.2	1.2.2	1.2	1.0	3
2.0	2.0	2.0	2.0	2.0	0
3.0	3.0	3.0	3.0	3.0	0

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 59–8.

Table 59–8 Column-Flattened Result Set with Data Security and DescendantOf Operator

C0	C1	C2	C3	C4	Distance
1.0	1.0	1.0	1.0	1.0	0
1.1	**1.1**	**1.1**	**1.1**	**1.0**	**1**
1.2	1.2	1.2	1.2	1.0	1
1.2.1	1.2.1	1.2.1	1.2	1.0	2
1.2.1.1	1.2.1.1	1.2.1	1.2	1.0	3
1.2.1.1.1	1.2.1.1	1.2.1	1.2	1.0	4
1.2.2	**1.2.2**	**1.2.2**	**1.2**	**1.0**	**2**
1.2.2.1	**1.2.2.1**	**1.2.2**	**1.2**	**1.0**	**3**
1.2.2.2	**1.2.2.2**	**1.2.2**	**1.2**	**1.0**	**3**
2.0	2.0	2.0	2.0	2.0	0
3.0	3.0	3.0	3.0	3.0	0

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.

59.8.6.1 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.

59.9 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:

- Section 22.12, "Preparing Descriptive Flexfield Business Components for Oracle Business Intelligence"
- Section 23.4.3, "How to Prepare Key Flexfield Business Components for Oracle Business Intelligence"

59.10 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

59.10.1 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:

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.

59.10.2 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:

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.

59.11 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.

60

Implementing ADF Desktop Integration

This chapter describes how to combine third party desktop applications with Oracle Fusion web applications.

The chapter includes these sections:

- Section 60.1, "Oracle Application Development Framework Desktop Integration Standards and Guidelines"
- Section 60.2, "Skinning Excel ADF Desktop Integration Workbooks"
- Section 60.3, "Configuring the WebLogic Server Frontend"

60.1 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. As a result, 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 *Oracle Fusion Middleware Desktop Integration Developer's Guide for Oracle Application Development Framework*.

For information about using ADF Desktop Integration technology with flexfields, see Section 22.14, "Accessing Descriptive Flexfields from an ADF Desktop Integration Excel Workbook" and Section 23.4.5, "How to Access Key Flexfields from an ADF Desktop Integration Excel Workbook".

Important: 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.

60.1.1 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>
|   |   |               |-- <LBA Core>
```

```
|   |   |                   |-- di
|   |   |                       |-- excel
```

Example 60–1 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 60–1 Directory Structure in WIndows

```
D:\FinDashboardPrototype\gl\components\journals\desktopJournalEntry\di\adfmsrc\ora
cle\apps\financials\generalLedger\journals\desktopJournalEntry\di\pageDefs

D:\FinDashboardPrototype\gl\components\journals\desktopJournalEntry\di\public_
html\oracle\apps\financials\generalLedger\journals\desktopJournalEntry\di\[excel|p
age]

D:\FinDashboardPrototype\gl\components\journals\desktopJournalEntry\di\src\oracle\
apps\financials\generalLedger\journals\desktopJournalEntry\di\bean
```

Example 60–2 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 60–2 Directory Structure in Source Control

```
scs/gl/components/journals/desktopJournalEntry/di/adfmsrc/oracle/apps/financials/g
eneralLedger/journals/desktopJournalEntry/di/pageDefs

scs/gl/components/journals/desktopJournalEntry/di/public_
html/oracle/apps/financials/generalLedger/journals/desktopJournalEntry/di/[excel|p
age]

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/gener
alLedger/journals/desktopJournalEntry/di/bean
```

60.1.2 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.

60.1.3 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 60–1.

Table 60–1 Tag Properties Exposed on the Declarative Component

Component Property	Data Type	Description
`closeWindowBinding`	String	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="#{SubmissionOptionsBean.dialogResult}"` 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.
`downloadAfterUpload`	Boolean	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 needs to be specified only if you are using the dialog as a custom upload dialog. Example: `downloadAfterUpload="false"`
`abortUploadOnFailure`	Boolean	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 needs to be specified only if you are using the dialog as a custom upload dialog. Example: `abortUploadOnFailure="false"`

Note: If you are using the page as a simple dialog (a basic web picker that only needs `ADFdi_CloseWindow`), you only need to specify the `closeWindowBinding`. Do not specify values for the `downloadAfterUpload` and `abortUploadOnFailure` properties, which only need to be specified for a custom upload dialog.

60.1.3.1 Adding 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.
   - In the Resource Palette (**View > Resource Palette**), right-click **File System** and select **New 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`.

Once 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:

   ```
   <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:f="http://java.sun.com/jsf/core"
   xmlns:af="http://xmlns.oracle.com/adf/faces/rich"
   xmlns:c="http://java.sun.com/jsp/jstl/core"
   xmlns:di="http://xmlns.oracle.com/apps/financials/common/publicDi/component/diC
   omponents">
   ```

   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:

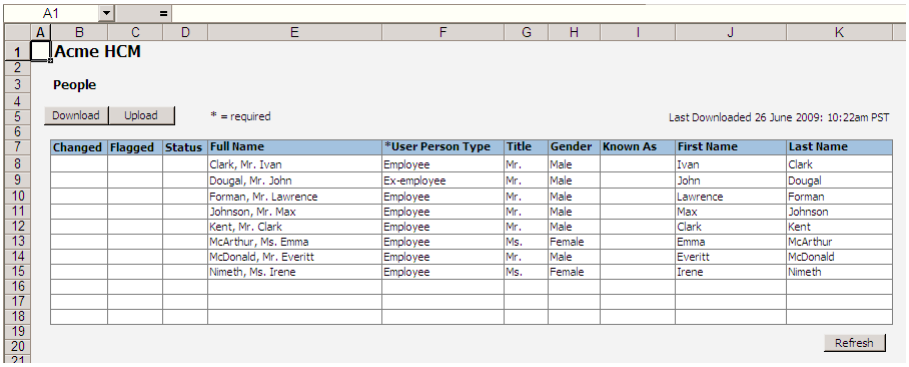
   ```
   <di:dialogAttributes id="da1"
   abortUploadOnFailure="false"
   downloadAfterUpload="false"
   closeWindowBinding="#{SubmissionOptionsBean.dialogResult}"/>
   ```

You are now ready to run your project.

60.2 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 60–1 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 60–1 Example of ADF Desktop Integration Look and Feel in Excel



Changed	Flagged	Status	Full Name	*User Person Type	Title	Gender	Known As	First Name	Last Name
			Clark, Mr. Ivan	Employee	Mr.	Male		Ivan	Clark
			Dougal, Mr. John	Ex-employee	Mr.	Male		John	Dougal
			Forman, Mr. Lawrence	Employee	Mr.	Male		Lawrence	Forman
			Johnson, Mr. Max	Employee	Mr.	Male		Max	Johnson
			Kent, Mr. Clark	Employee	Mr.	Male		Clark	Kent
			McArthur, Ms. Emma	Employee	Ms.	Female		Emma	McArthur
			McDonald, Mr. Everitt	Employee	Mr.	Male		Everitt	McDonald
			Nimeth, Ms. Irene	Employee	Ms.	Female		Irene	Nimeth

60.3 Configuring 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.

61

Creating 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 also provides information about how to perform runtime customizations.

This chapter includes the following sections:

61.1 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.

A customized application contains a base application and one or more layers of customized metadata content. MDS stores the customized metadata objects in a metadata repository and retrieves them at run time to reveal the customized application. When a customized application is launched, the customized content is applied over the base content.

For more information about MDS repositories (database and file-based) and metadata archives (MAR), see the "Managing the Metadata Repository" chapter in the *Oracle*

Fusion Applications Administrator's Guide. For more information about customization, see the "Customizing Applications with MDS" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. For information about the ways in which developers and end users can customize the application, see the "Customizing and Extending Oracle Fusion Applications" chapter in the *Oracle Fusion Applications Extensibility Guide*.

You must perform the following tasks in the order shown in your JDeveloper application workspace to ensure that your pages can be customized by customers. You must complete all the non-optional tasks.

1. Set project properties to enable user and seeded customizations.
2. Configure the persistence change manager parameter to use the composer change manger.
3. Define the customization layers and their order of precedence.
4. Authorize runtime customization of pages and task flows.
5. (Optional) Restrict customizations for specific components on a page.
6. Set up resource bundles for translation.
7. Enable application pages for runtime customization.
8. (Optional) Enable user customization of the user interface (UI) shell template.
9. Create an IDE connection to your database (in addition to the application connection) to enable access to layer values.

Once the application is configured to enable customizations, customizations can be created in two different ways:

- At design time using JDeveloper, also known as design-time customization.

  Design-time customizations that are created and shipped with Oracle Fusion Applications are known as seeded customizations.

- At runtime using Oracle Composer or Oracle Application Development Framework (ADF) implicit customizations, also known as runtime customizations.

End users are also able to customize the navigator menu that displays in the UI Shell global area.

61.2 Setting Project Properties to Enable User and Seeded Customizations

To allow JDeveloper-based and Oracle Composer-based customizations for your application, you must enable user and seeded customizations.

61.2.1 How to Set Project Properties to Enable User and Seeded Customizations

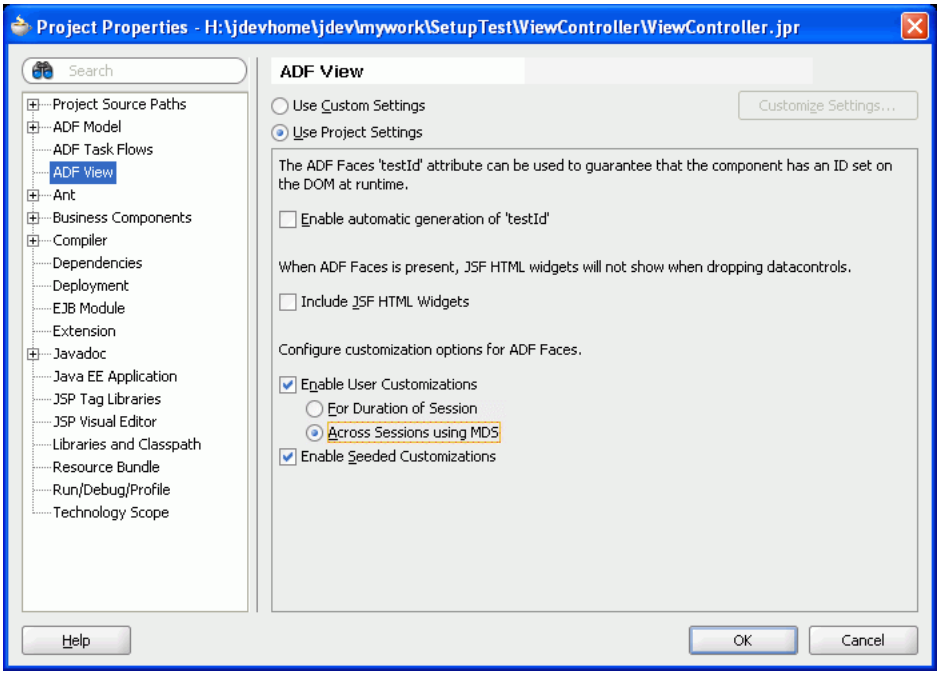
To enable user customization, you must configure the view project to allow user customizations, and you must configure the view project to persist the customized metadata objects to a MDS repository so that the objects are available across sessions. You must also enable seeded customizations so that 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, with the exception of 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 61–1.
4. Select **Enable Seeded Customizations**.

Figure 61–1 Project Properties — ADF View



5. Click **OK**.

61.3 Configuring the Persistence Change Manager

Before you can define customization layers, you must configure the persistence change manager parameter to use the composer change manger

61.3.1 How to Configure the Persistence Change Manager

When you enabled user customizations across sessions using MDS by completing the procedure in Section 61.2, "Setting 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 using MDS as described in Section 61.2, "Setting Project Properties to Enable User and Seeded Customizations."

To configure the persistence change manager:

1. Expand the **WEB-INF** node for your view project, right-click the **web.xml** node and choose **Open**.
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 61–1.

Note: Filters must be configured in the following order.

1. `JpsFilter`
2. `ApplSessionFilter`
3. `WebCenterComposerFilter`
4. `ADFBindingFilter`

Example 61–1 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>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>REQUEST</dispatcher>
  </filter-mapping>
....
```

4. Enable sessions for the JSPX pages and task flows that you create in your view project as described in Section 47.2.1, "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.

61.4 Defining the Customization Layers

A customizable application can have multiple customization layers, such as Global, Site, and User. You need to define the application's customization layers and their order of precedence.

For information about customization layers, see the "Understanding Customization Layers" section in the *Oracle Fusion Applications Extensibility Guide*.

61.5 Authorizing Runtime Customization of Pages and Task Flows

When a user opens an application page in a browser, the page opens in view mode. Additionally, Oracle Composer provides access to another mode that enables users with appropriate privileges to customize and personalize application pages and to save the changes as customizations that are available for all users accessing the page. Users can also be granted the privilege to edit a task flow at runtime.

For information about using Oracle Composer to customize pages, see the "Customizing Existing Pages" chapter of the *Oracle Fusion Applications Extensibility Guide*.

61.5.1 How to Authorize Runtime Customization of Pages and Task Flows

You modify the resource grants in the `jazn-data.xml` file to authorize runtime customization of pages and task flows.

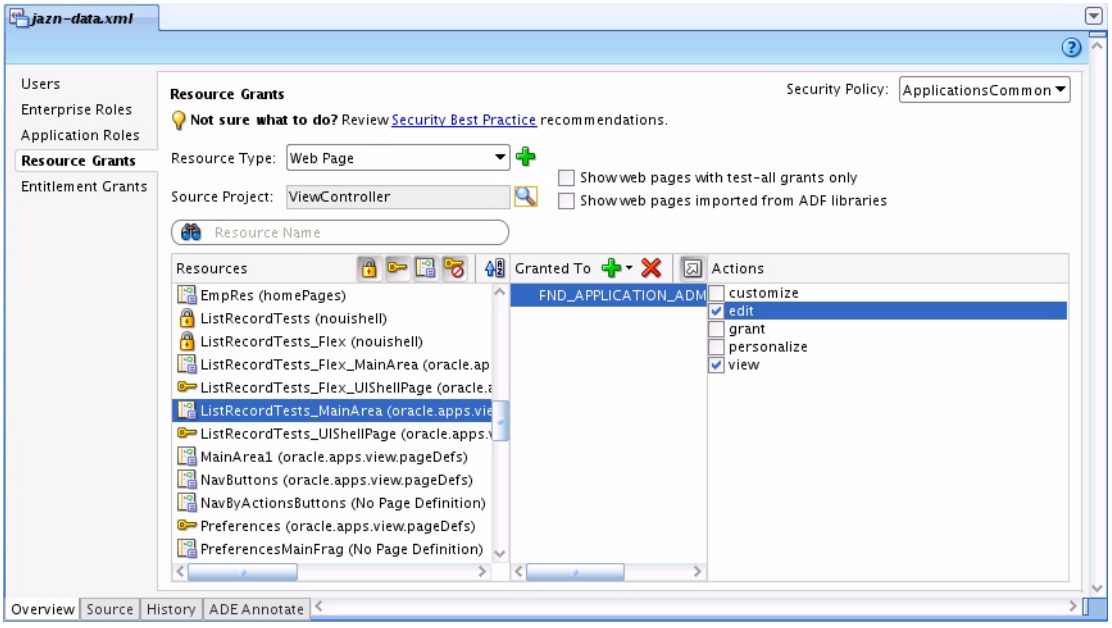
To authorize pages and task flows for runtime customizations:

1. Open the `jazn-data.xml` file, which is located in **Application Resources > Descriptors > META-INF**.

   Tip: If the `jazn-data.xml` file does not exist, you can create it by right-clicking the **META-INF** folder, selecting **New Oracle Deployment Descriptor**, selecting `jazn-data.xml`, and clicking **Finish**.

2. In the overview editor, click the **Resource Grants** navigation tab, as shown in Figure 61–2.

Figure 61–2 Resource Grants Tab in jazn-data.xml Overview



3. Select **Web Page** from the **Resource Type** dropdown list.
4. Set the **Source Project** to `ViewController`.
5. For each Oracle Composer enabled web page, select the **Customize and Personalize** action for each role for which you want to enable page customization.
6. Select **Task Flow** from the **Resource Type** dropdown list.
7. For each task flow that Oracle Composer can customize in your application, select the **Customize** and **Personalize** actions for each role for which you want to enable task flow customization.
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**, **grant**, and **personalize Actions**.

For more information, see the "Implementing Task Flow Security" section in the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

61.6 Restricting Customization for a Specific Component on a Page

Customization is enabled for all components on a page by default. However, there might be situations where you want to prevent customization for some of the components on a page.

61.6.1 How to Restrict Customization for a Specific Component on a Page

You can specify at the component level whether customizations for the component are permitted at runtime and who is permitted to customize that component.

How to restrict customization for a page component:

1. In the Application Navigator, select the page or fragment for which you want to edit customization properties.
2. In the Structure window, select the component for which you wish to restrict customization.
3. If the Property Inspector is not open, choose **Property Inspector** from the **View** menu.
4. In the Property Inspector, expand the **Customization** node.
5. Set the appropriate customization attribute.
   - To disable customization of a component at runtime, 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 *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

61.7 Configuring Runtime Resource String Editing for Customizations

The resource string editor enables 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.

61.7.1 How to Configure Runtime Resource String Editing for Customizations

You must configure your application to enable runtime resource string editing of translated values for customizations.

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. In the **File Name** field in the Select Resource Bundle dialog, type `oracle.adf.view.page.editor.resource.ComposerOverrideBundle.xlf`, and click **Open**.
5. Select the **Overridden** check box.
6. Click **OK** to save your changes.
7. Open the `adf-config.xml` file, which is located in the **Application Resources > Descriptors > ADF META_INF** folder.
8. Add the `resource-string-editor` element shown in Example 61–2 to the `page-editor-config` section to enable resource string editing.

Example 61–2 Configuration to Turn On Resource Picker

```
<pe:page-editor-config>
  ...
```

```
  <resource-string-editor>
    <enabled>
      #{GlobalAreaBackingBean.tipLayerNonUser}
    </enabled>
  </resource-string-editor>
</pe:page-editor-config>
```

This setting only enables resource string editing if the changes are customizations and not user personalizations, as user personalizations do not need to be translated.

For more information about configuring the runtime resource string editor, see the "Configuring Runtime Resource String Editing" section in the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

61.8 Enabling Pages for Runtime Customization

Pages can be customized at runtime using Oracle Composer.

61.8.1 How to Enable Pages for Runtime Customization

You must perform the following steps to enable runtime customization of a web page using Oracle Composer:

1. Add the Oracle Composer technology scope to your project.
2. Prepare your page for end-user personalizations.
3. Ensure that the customizable pages have page definition files.
4. Make pages runtime editable by adding Oracle Composer components to the pages.
5. If you are enabling the runtime addition of content, set up a resource catalog.
6. (Optional) Set up a default catalog definition file to facilitate testing.
7. (Optional) Persist elements and properties across sessions outside of Oracle Composer.

61.8.1.1 Adding Oracle Composer Technology Scope to Your Project

You must add the Oracle Composer technology scope in order to access the technologies for consuming Oracle Composer components and enabling runtime customization.

To add Oracle Composer technology scope:

1. Right-click your view project and select **Project Properties**.
2. Select **Technology Scope**.
3. Add the **Oracle Composer** technology scope to your project and click **OK**.

61.8.1.2 Preparing Your Page for End-User Personalizations

You 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 pane, select the `af:pageTemplate` node.
3. In the Property Inspector, select **true** from the **isPersonalizableInComposer** dropdown list.

61.8.1.3 Ensuring 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.

61.8.1.4 Making a JSPX Document Editable at Runtime

To make a JSPX document editable at runtime, you add Oracle 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 Composer technology scope, which you added when you completed the steps in Section 61.8.1.1, "Adding Oracle Composer Technology Scope to Your Project," and which are available from the Oracle Composer page in the Component Palette.

For more information about the Panel Customizable and Layout Customizable components, see the "Oracle Composer Components" section in the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

61.8.1.5 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 *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

61.8.1.6 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 default catalog definition file `default-catalog.xml` to the newly created `<application-root>/ViewController/src/oracle/adf/rc/metadata` directory.

> **Caution:** The Oracle WebCenter default catalog should only be used for testing purposes. You must create your own catalog for production purposes.

61.8.1.7 Configuring the Persistence of Implicit Runtime Customizations

Certain ADF Faces components have attributes that can be saved during a user session. For example, whether the user left a panel box component expanded or collapsed. This type of change is referred to as implicit customization.

In Section 61.2, "Setting Project Properties to Enable User and Seeded Customizations," you configured the application to persist user customizations across sessions. This enables you to configure attributes to be saved across sessions. For information about how to configure the persistence of component attribute values, see the "Allowing User Customizations at Runtime" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

61.9 Enabling 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 Oracle Composer to customize UI Shell pages, as described in the "Customizing Existing Pages" chapter in the *Oracle Fusion Applications Extensibility Guide*.

61.9.1 How to Enable User Customization of the UI Shell Template

To enable users to customize the UI shell template, you add a link or button to the page fragment that launches the UI Shell template that allows the UI Shell template to be customized from Oracle Composer.

To add a link to a page fragment:

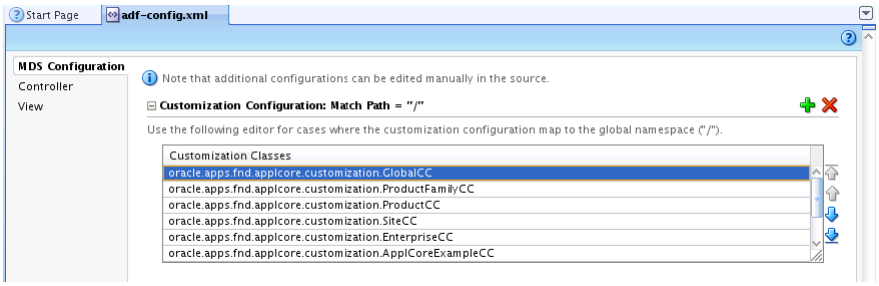
1. From JDeveloper, open the page fragment (JSFF) from where the UI Shell can be edited.
2. In the Application Navigator, expand the **Data Controls** hierarchy to locate and expand the **FndUIShellController**.
3. Drag and drop the **customizeUIShellTemplate** operation onto the page, and choose **Create > 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 61–3.

Example 61–3 Sample custClass List

```
oracle.apps.fnd.applcore.customization.GlobalCC,oracle.apps.fnd.applcore.customiza
tion.SiteCC
```

Each of the customization classes supplied in the list must be valid and configured in the `adf-config.xml` file, as shown in Figure 61–3.

Figure 61–3 adf-config.xml — Customization Classes



If any of the classes cannot be instantiated, or if they are not pre-configured in the `adf-config.xml` file, an exception is thrown at runtime.

The last customization class specified is the *tip* customization layer and the modifications to the UI Shell is written to this layer. In Example 61–3, 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.

5. Click OK.
6. Open the page definition file for the page fragment.
7. In the editor window, click the **Source** tab.
8. Add the `<methodAction>` element shown in Example 61–4 to the `<bindings>` element. If you are enabling the customization of a UI Shell page other than `TemplateCustomizationUIShell`, change the `viewID` value to the view ID for that page.

Example 61–4 custNavigate methodAction Binding

```
  <bindings>
    <methodAction id="custNavigate" RequiresUpdateModel="true"
                  Action="invokeMethod" MethodName="navigate"
                  IsViewObjectMethod="false" DataControl="FndUIShellController"
                  InstanceName="FndUIShellController.dataProvider"
                  ReturnName=
"FndUIShellController.methodResults.navigate_FndUIShellController_dataProvider_
navigate_result">
      <NamedData NDName="viewId" NDValue="TemplateCustomizationUIShell"
                 NDType="java.lang.String"/>
      <NamedData NDName="webApp" NDType="java.lang.String"/>
      <NamedData NDName="pageParametersList" NDType="java.lang.String"/>
      <NamedData NDName="navTaskFlowId" NDType="java.lang.String"/>
      <NamedData NDName="navTaskKeyList" NDType="java.lang.String"/>
      <NamedData NDName="navTaskParametersList" NDType="java.lang.String"/>
      <NamedData NDName="navTaskLabel" NDType="java.lang.String"/>
      <NamedData NDName="methodParameters"
  NDType="oracle.apps.fnd.applcore.patterns.uishell.ui.bean.FndMethodParameters"/>
    </methodAction>
  </bindings>
```

9. If you are enabling the customization of a UI Shell page in another application, provide the values for the `viewId`, `webApp`, and `pageParametersList`, as shown in Example 61–5. For more information about these values, see Table 14–15.

Example 61–5 Example of viewId, webApp, and pageParametersList Values

```
<NamedData NDName="viewId" NDValue="specialUIShellPage"
           NDType="java.lang.String"/>
<NamedData NDName="webApp" NDValue="commonApp"
           NDType="java.lang.String"/>
<NamedData NDName="pageParametersList" NDValue="taskflow1"
           NDType="java.lang.String"/>
```

61.10 Creating 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 IDE level connection.

The IDE level connection is required when implementing design-time customizations from JDeveloper, as described in the "Using JDeveloper for Customizations" chapter in the *Oracle Fusion Applications Extensibility Guide*.

61.10.1 How to Create a Database Connection at the IDE Level

You create the IDE level connection from the Database Navigator.

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.

61.11 Implementing Design-Time Customizations from JDeveloper

While Oracle Composer enables user interface customizations, other customizations, such as changes to the model or task flow roles, must be done from JDeveloper. For information about implementing design-time customizations from JDeveloper, see the "Using JDeveloper for Customizations" chapter in the *Oracle Fusion Applications Extensibility Guide*.

61.12 Implementing Runtime Customizations

You can implement the following types of runtime customizations:

- Implicit runtime customizations: These are runtime customizations that are based on the user's behavior, such as saving an entered value or remembering whether the user last expanded or collapsed a panel.

For information about implementing implicit runtime customizations, see the "Allowing User Customizations at Runtime" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

For information about saving user customizations across sessions, see Section 61.8.1.7, "Configuring the Persistence of Implicit Runtime Customizations."

- Explicit runtime customizations: These are customizations that the user explicitly implements by running a customization tool. For information about implementing explicit runtime customizations, see the *Oracle Fusion Applications Extensibility Guide*.

For information about building editable pages, see the "Allowing User Customizations at Runtime" chapter in the *Oracle Fusion Middleware Developer's Guide for Oracle WebCenter*.

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).

61.13 Customizing the Navigator Menu

The *navigator menu* is the global menu that displays in the UI Shell global area, and is displayed when you click Navigator buttons or links in the UI Shell, as shown in Figure 61–4.

Figure 61–4 Navigator Menu Example

A user with the required privileges can perform the following tasks to customize the menu at the Site level:

- Add, delete, or edit a group
- Add, delete, or edit an item
- Specify navigation
  - Navigate to a taskflow in the application
  - Navigate to a URL
- Hide a group or an item or show a hidden group or item

Note: You cannot customize the Home menu or the Preferences menu.

For more information, see the "Customizing the Navigator Menu" chapter in the *Oracle Fusion Applications Extensibility Guide*.

Part IX

Appendices

The appendices provide information about how to work with the Oracle Fusion application taxonomy, and a reference for the commands available for the Oracle Enterprise Crawl and Search (ECSF) Command Line Administration Utility.

The *Oracle Fusion Applications taxonomy* organizes the components and functions of Oracle Fusion Applications into a hierarchical structure. Every Oracle Fusion development artifact or file is tagged with an owning functional component. Components are grouped hierarchically into larger units, such as more general components, products and product families.

ECSF Command Line Administration Utilities 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.

This part contains the following appendices:

A

Working 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:

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.

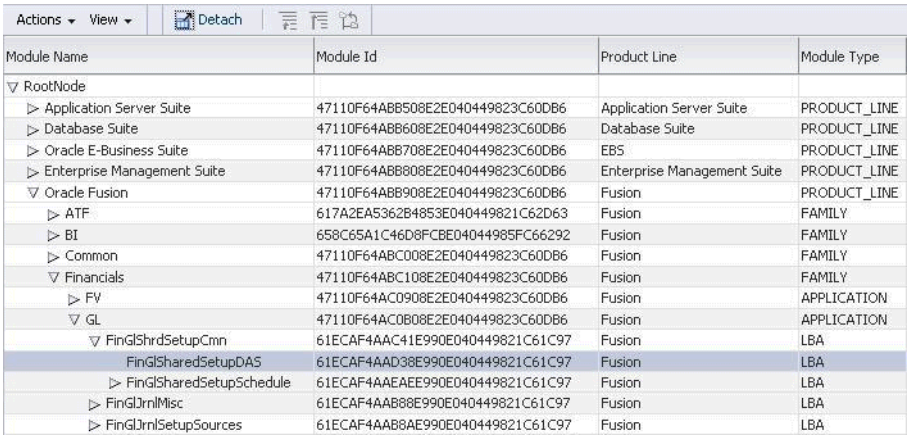
A.1 Introduction to the Oracle Fusion Application Taxonomy

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 A–1:

Figure A–1 Taxonomy UI Hierarchy Example



Module Name	Module Id	Product Line	Module Type
▽ RootNode			
▷ Application Server Suite	47110F64ABB508E2E040449823C60DB6	Application Server Suite	PRODUCT_LINE
▷ Database Suite	47110F64ABB608E2E040449823C60DB6	Database Suite	PRODUCT_LINE
▷ Oracle E-Business Suite	47110F64ABB708E2E040449823C60DB6	EBS	PRODUCT_LINE
▷ Enterprise Management Suite	47110F64ABB808E2E040449823C60DB6	Enterprise Management Suite	PRODUCT_LINE
▽ Oracle Fusion	47110F64ABB908E2E040449823C60DB6	Fusion	PRODUCT_LINE
▷ ATF	617A2EA5362B4853E040449821C62D63	Fusion	FAMILY
▷ BI	658C65A1C46D8FCBE04044985FC66292	Fusion	FAMILY
▷ Common	47110F64ABC008E2E040449823C60DB6	Fusion	FAMILY
▽ Financials	47110F64ABC108E2E040449823C60DB6	Fusion	FAMILY
▷ FV	47110F64AC0908E2E040449823C60DB6	Fusion	APPLICATION
▽ GL	47110F64AC0B08E2E040449823C60DB6	Fusion	APPLICATION
▽ FinGlShrdSetupCmn	61ECAF4AAC41E990E040449821C61C97	Fusion	LBA
FinGlSharedSetupDAS	61ECAF4AAD38E990E040449821C61C97	Fusion	LBA
▷ FinGlSharedSetupSchedule	61ECAF4AAEAEE990E040449821C61C97	Fusion	LBA
▷ FinGlJrnlMisc	61ECAF4AAB88E990E040449821C61C97	Fusion	LBA
▷ FinGlJrnlSetupSources	61ECAF4AAB8AE990E040449821C61C97	Fusion	LBA

A.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.

A.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

A.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.

A.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.

A.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.

A.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.

A.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.

A.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 A–1 lists the task flow and its parameters.

Table A–1 Taxonomy Task Flow and Parameters

Task Flow Name	Task Flow XML	Patterns Passed	Behavior	Comments
Manage Taxonomy Hierarchy	`/WEB-INF/oracle/apps/fnd/applcore/taxonomy/ui/taskflow/ViewDeliveryHierarchy.xml#ViewDeliveryHierarchy`	[pageTitle]	The Manage Taxonomy Hierarchy accepts the optional parameter [pageTitle] and navigates to the Taxonomy Delivery Hierarchy page.	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.

A.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?).

A.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 A–2:

Table A–2 Who Columns

Column Name	Type	Null?
CREATED_BY	VARCHAR2(64)	Not Null
CREATION_DATE	TIMESTAMP	Not Null
LAST_UPDATED_BY	VARCHAR2(64)	Not Null
LAST_UPDATE_DATE	TIMESTAMP	Not Null
LAST_UPDATE_ LOGIN	VARCHAR2(32)	

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 A–3. You do not need to add extended Who columns if the table does not already have them.

Table A–3 Extended Who Columns

Column Name	Type	Null?
REQUEST_ID	NUMBER(20)	NULL
JOB_DEFINITION_ NAME	VARCHAR2(100)	NULL
JOB_DEFINITION_ PACKAGE	VARCHAR2(900)	NULL

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).

A.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.

A.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 `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`.
- **ApplTaxonomySeedDataPVO** — Used for extracting taxonomy seed data. 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`.
- **ApplTaxonomyApplicationPVO**

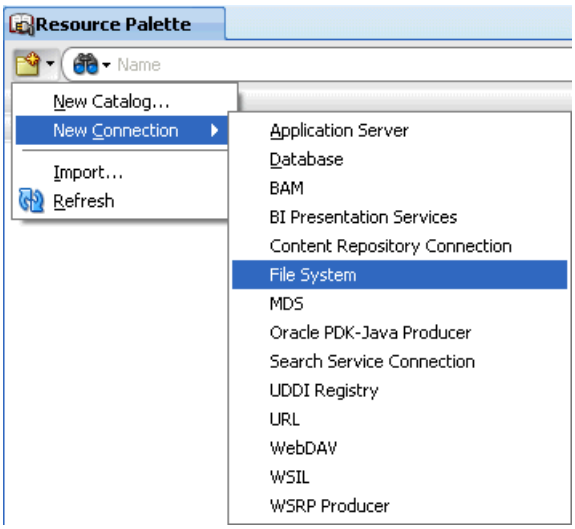
- **ProductLineCriteria**: where `ApplTaxonomyEO.PRODUCT_LINE=:bProductLine` where `:bProductLine` is a named bind variable with a default value of `1` – the Oracle Fusion Productline.
- **IsSeedDataAllowedCriteria**: where `ApplTaxonomyEO.IS_SEED_DATA_ALLOWED=:bIsSeedDataAllowed`, where `:bIsSeedDataAllowed` is a named bind variable with a default value of `Y`.

A.2.3.1 Accessing the Entity and View Objects

To access the Entity and View objects, and other taxonomy components, create a new **File System Connection** to `adflib`:

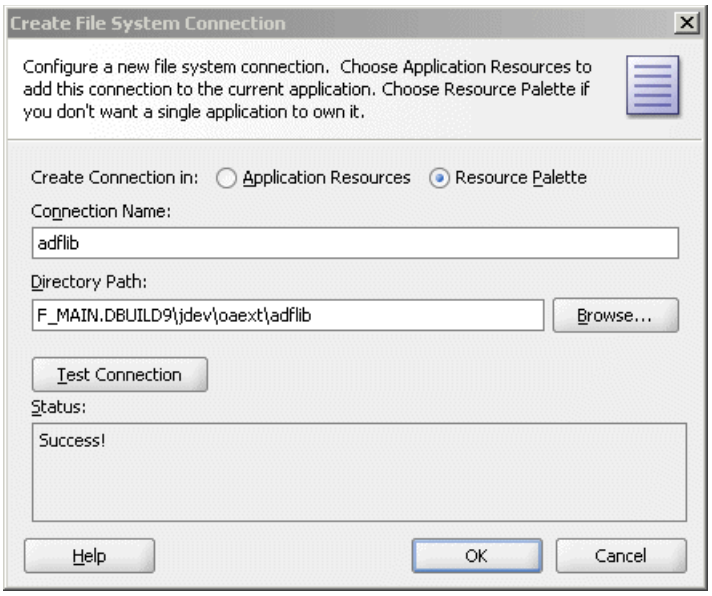
1. In the **Resource Palette**, open the folder icon and select **New Connection > File System** as shown in Figure A–2:

Figure A–2 Creating a New File System Connection



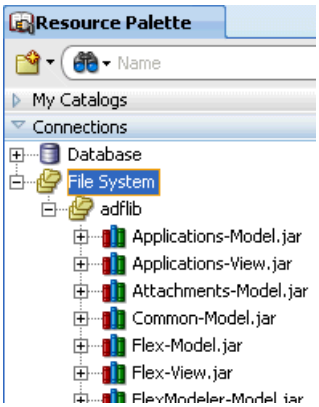
2. Configure the new connection so it resembles the example in Figure A–3:

Figure A–3 Configuring a New File System Connection



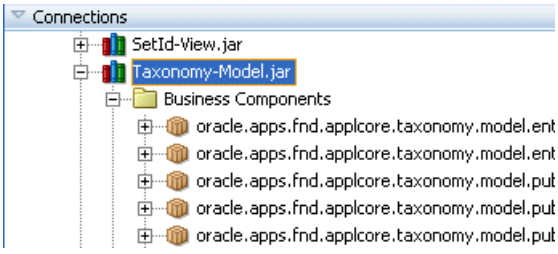
3. The new connection will display in the **Connections** tree, shown in Figure A–4:

Figure A–4 New File System Connections Tree



4. The Entity and View objects are located in **Taxonomy-Model.jar > Business Components**, shown in Figure A–5:

Figure A–5 Locating the Entity and View Objects in the Connections Tree



A.2.4 How to Use Exposed Service Methods

Notes:

- 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 moduleId, this returns the taxonomy node:

  ```
  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.TAXONO
MY_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.

A.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 `getChildApplTaxonomyFullDeliveryVO()`, as shown in Example A–1:

Example A–1 Obtaining the Children of the Current Node

```
RowIterator children = row.getChildApplTaxonomyFullDeliveryVO();
while(children.hasNext())
{
    ApplTaxonomyFullDeliveryVORowImpl childRow =
        (ApplTaxonomyFullDeliveryVORowImpl)children.next();
    // Your code here.
}
ApplTaxonomyFullDeliveryVORowImpl parentRow =  (ApplTaxonomyFullDeliveryVORowImpl)
row.getParentApplTaxonomyFullDeliveryVO();
```

To obtain the parent of the current node, call `getParentApplTaxonomyFullDeliveryVO()`, as shown in Example A–2.

Example A–2 Obtaining the Parent of the Current Row

```
ApplTaxonomyFullDeliveryVORowImpl parentRow =
   (ApplTaxonomyFullDeliveryVORowImpl) row.getParentApplTaxonomyFullDeliveryVO();
```

A.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.

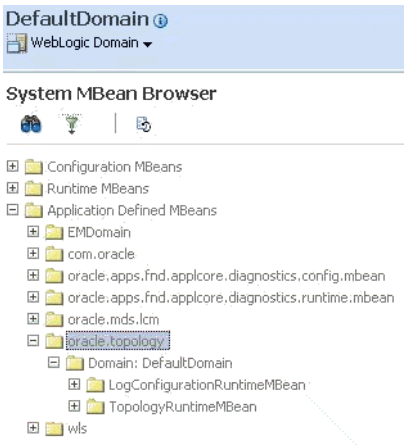
Types of Taxonomy MBeans

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 A–6.

Figure A–6 MBeans Viewed from Enterprise Manager



Topology MBean

Topology MBean Details:

```
MBean Name oracle.topology:name=Topology,type=TopologyRuntimeMBean
Description Applications Core Topology MBean that provides the information about
overall Oracle Fusion Topology
```

Attributes exposed by Topology MBeans are shown in Table A–4.

Table A–4 Attributes Exposed by Topology MBeans

Name	Description	Access
AllProductFamilyAndDomains	Gets all domains and product families of an Oracle Fusion instance.	R
ConfigMBean	If true, it indicates that this is a Config MBean.	R
CurrentDomain	Gets the current domain.	R
CurrentPillarInfo	Get the current pillar.	R
CurrentPillarInstanceInfo	Returns all information about current PillarInstanceName.	R
CurrentProductFamily	Gets the current product family module key list.	R
CurrentProductFamilyInfo	Gets the current product family list information.	R
eventProvider	If true, it indicates that this MBean is an event provider as defined by JSR-77.	R
eventTypes	All the event types emitted by this MBean.	R
ListOfProducts	Gets the list of products for the current product family.	R
objectName	The MBean's unique JMX name.	R
PillarDBInfo	Get the database information for a particular pillar.	R

Table A–4 (Cont.) Attributes Exposed by Topology MBeans

Name	Description	Access
Pillars	Get all the pillars.	R
ProductFromEachProductFamily	Gets list of products for each product family.	R
ReadOnly	If true, this MBean is read-only.	R
RestartNeeded	Indicates whether a restart is needed.	R
stateManageable	If true, it indicates that this MBean provides state management capabilities as defined by JSR-77.	R
statisticsProvider	If true, it indicates that this MBean in a statistic provider as defined by JSR-77.	R
SystemMBean	If true, it indicates that this MBean is a System MBean.	R

The operations exposed by Topology MBeans are shown in Table A–5.

Table A–5 Operations Exposed by Topology MBeans

Name	Description	Parameters	Return Type
getAllDeployedAppsInfo	Complete information on the deployed application for a particular product.	1	Array of javax.management.openmbean.TabularData
getAllEnterpriseAppsInfo	Gets the list of application information for a given product.	2	Array of javax.management.openmbean.TabularData
getAppListFromDeployedDomain	Get the application information from the deployed domain name.	1	Array of javax.management.openmbean.TabularData
getDependentApps	Returns dependent applications information on an application from its AppShortName.	1	Array of javax.management.openmbean.TabularData
getDependentMWComponents	Gets Dependent MW Components for the AppShortName.	1	Array of javax.management.openmbean.TabularData
getDeployedAppInfo	Complete information on the deployed application given for an Application Short Name.	1	Array of javax.management.openmbean.TabularData
getDeployedDomainFromLogicalDomain	Gets DeployedDomain information from the LogicalDomain name.	1	Array of javax.management.openmbean.TabularData
getDeployedDomainFromPillar	Get the deployed domain information of a particular type for a pillar.	2	Array of javax.management.openmbean.TabularData
getDeployedDomainInfo	Returns a map of domain information for a given domain name.	1	javax.management.openmbean.TabularData
getDeployedDomainByCompositeName	Get the deployed domain list for a given composite name.	1	Array of javax.management.openmbean.TabularData
getDeployedDomainsByEnvironment	Returns the list of domains for a particular environment. If the EnvironmentShortName is null, it will return all the deployed domains.	1	javax.management.openmbean.TabularData

Table A–5 (Cont.) Operations Exposed by Topology MBeans

Name	Description	Parameters	Return Type
getDomainnames	Gets the list of domain names for a given application short name.	1	Array of java.lang.String
getEndPointInfo	Gets the external and internal end points for a given application short name.	1	Array of javax.management.openmbean.TabularData
getEndPointInfoFromModule	Gets the domain external and internal domain end points for a Logical Module Name.	1	Array of javax.management.openmbean.TabularData
getEnterpriseAppInfo	Gets the list of application information for a given application short name.	1	javax.management.openmbean.TabularData
getEssApplicationInfo	Gets ESS application information for a given product family module id.	1	javax.management.openmbean.TabularData
getLbasInfo	Gets the list of LBA information for the given product family.	1	Array of javax.management.openmbean.TabularData
getListOfDeployedApps	Gets the list of deployed applications for a particular product.	1	Array of java.lang.String
getListOfDomains	Gets the list of domain names for a given product family	1	Array of java.lang.String
getListOfEnterpriseApps	Gets the list of applications for a given product.	2	Array of java.lang.String
getListOfLbas	Gets the list of LBA (module names) for the given product family.	1	Array of java.lang.String

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 A–6.

Table A–6 Attributes Exposed by Log Configuration MBeans

Name	Description	Access
ConfigMBean	If true, it indicates that this is a Config MBean.	R
eventProvider	If true, it indicates that this MBean is an event provider as defined by JSR-77.	R
eventTypes	All the event types emitted by this MBean.	R
LogConfigInformation	Gets the log configuration information at Site level.	R
objectname	The MBean's unique JMX name.	R
ReadOnly	If true, this MBean is read-only.	R
RestartNeeded	Indicates whether a restart is needed.	R
stateManageable	If true, it indicates that this MBean provides State Management capabilities as defined by JSR-77.	R
statisticsProvider	If true, it indicates that this MBean in a statistic provider as defined by JSR-77.	R
SystemMBean	If true, it indicates that this MBean is a System MBean.	R

Operations exposed by Log Configuration MBeans are shown in Table A–7.

Table A–7 Operations Exposed by Log Configuration MBeans

Name	Description	Parameters	Return Type
addUserLogConfig	Adds the log configuration information for a particular user.	8	boolean
deleteUserLogConfig	Delete the log configuration information for a particular user.	1	void
editUserLogConfig	Edit the log configuration information for a particular user.	8	boolean
getUserInfo	Gets the user information (the GUID) for users either having or not having the log configuration information.	2	javax.management.openmbean.TabularData
getUserLogConfigInformation	Gets the log configuration information for a particular user.	1	Array of javax.management.openmbean.TabularData
updateLogConfigInformation	Update the log configuration information at Site level.	8	void

Sample Code to Invoke MBean APIs

Sample code is shown in Example A–3.

Example A–3 Sample Code to Invoke MBean APIs

```
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");
}
```

B

ECSF Command Line Administration Utility

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.

> **Note:** Administrators should use Fusion Applications Control for ECSF to manage the life cycle of searchable objects in the production environment.

Table B–1 shows the commands you can use to administer search. The commands appear in alphabetical order.

Table B–1 ECSF Command Line Administration Utility Commands

Command	Description
`activate object` *`ID`*	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. Only a searchable object that has been deployed can be activated. 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.
`add object` *`ID`* `to category` *`ID`*	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. 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.
`add object to category` *`ID`*	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.

Table B–1 (Cont.) ECSF Command Line Administration Utility Commands

Command	Description
`add object ID to schedule ID`	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.
`add object to schedule ID`	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.
`add unassigned object to instance`	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.
`add unassigned object ID to instance`	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.
`connect to database`	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.
`connect to database hostname port SID`	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.
`connect to database descriptor`	Creates the connection to a database using a database descriptor. Follow the prompts to enter a username and password, as well as field values.

Table B–1 (Cont.) ECSF Command Line Administration Utility Commands

Command	Description

`connect to database descriptor 'descriptor'`	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.

`connect to database service`	Creates the connection to a database using a service name. Follow the prompts to enter a username and password, as well as field values.
`connect to database service hostname port servicename`	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.
`connect to mbeanserver`	Creates the connection to an MBean server. Follow the prompts to enter a username and password, as well as field values.
`connect to mbeanserver hostname port`	Creates the connection to an MBean server. Specify the host name and port number. Follow the prompts to enter a username and password.
`create category`	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.
`create category set "fieldname"="value","fieldname"="value"...`	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, `"field name with \"escaped\" quotes".` 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.
`create instance`	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.
`create instance set "fieldname"="value","fieldname"="value"...`	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, `"field name with \"escaped\" quotes".` You must issue the command while not managing a search engine instance.

Table B–1 (Cont.) ECSF Command Line Administration Utility Commands

Command	Description
`create object`	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.
`create schedule`	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.
`create schedule set "`*`fieldname`*`="`*`value`*`","`*`fieldname`*`="`*`value`*`"...`	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, `"field name with \"escaped\" quotes"`. 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.
`create unassigned object`	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.
`deactivate object` *`ID`*	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.
`delete category` *`ID`*	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.
`delete external category` *`ID`*	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.
`delete instance` *`ID`*	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.

Table B–1 (Cont.) ECSF Command Line Administration Utility Commands

Command	Description
`delete object` *ID*	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.
`delete schedule` *ID*	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.
`delete unassigned object` *ID*	Removes the specified unassigned searchable object (not associated with a search engine instance) from the ECSF schema in the Oracle Fusion Applications database.
`deploy category` *ID*	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.
`deploy object` *ID*	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.
`deploy params for objects`	Collectively updates all deployed searchable objects with the latest search engine instance parameters.
`deploy params for object` *ID*	Updates the specified searchable object with the latest search engine instance parameters.
`deploy schedule` *ID*	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.
`disconnect`	Disconnects you from the current database or MBean server connection.
`exit`	Closes the ECSF Command Line Administration Utility.
`help`	Lists all the available `help` commands.
`help activate`	Lists the valid syntax for the `activate` command.
`help add`	Lists the valid syntax for the `add` commands.
`help category`	Lists the commands that can be used for search categories.
`help connect`	Lists the valid syntax for the `connect` commands
`help create category`	Lists the fields available for the `create category` commands.
`help create instance`	Lists the fields available for the `create instance` commands.
`help create object`	Lists the fields available for the `create object` command.
`help create schedule`	Lists the fields available for the `create schedule` command.

Table B–1 (Cont.) ECSF Command Line Administration Utility Commands

Command	Description
`help create unassigned object`	Lists the fields available for the `create unassigned object` command.
`help create schedule`	Lists the fields available for the `create schedule` commands.
`help deactivate`	Lists the valid syntax for the `deactivate object` command.
`help delete`	Lists the valid syntax for the `delete` commands.
`help deploy`	Lists the valid syntax for the `deploy` commands.
`help disconnect`	Lists the valid syntax for the `disconnect` command.
`help instance`	Lists the commands that can be used for search engine instances.
`help list`	Lists the valid syntax for the `list` commands.
`help manage`	Lists the valid syntax for the `manage` commands.
`help object`	Lists the commands that can be used for searchable objects.
`help param`	Lists the commands that can be used for search engine instance parameters.
`help remove`	Lists the valid syntax for the `remove` commands.
`help schedule`	Lists the commands that can be used for index schedules.
`help set`	Lists the valid syntax for the set commands.
`help showdetails`	Lists the valid syntax for the `showdetails` commands.
`help start`	Lists the valid syntax for the `start` command.
help stop	Lists the valid syntax for the `stop` command.
`help unassigned object`	Lists the commands that can be used for unassigned searchable objects.
`help undeploy`	Lists the valid syntax for the `undeploy` commands.
`help unmanage`	Lists the valid syntax for the `unmanage` commands.
`help update category`	Lists the fields available for the `update category` commands.
`help update instance`	Lists the fields available for the `update instance` commands.
`help update schedule`	Lists the fields available for the `update schedule` commands.
`help register object`	Lists the fields available for the `register object` commands.
`import external categories`	Lists one by one all the external categories of the search engine instance you are managing and prompts you to confirm whether or not 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.

Table B–1 (Cont.) ECSF Command Line Administration Utility Commands

Command	Description
`import external categories for instance ID`	Lists one by one all the external categories of the search engine instance you specify and prompts you to confirm whether or not 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.
`list categories`	Lists the search categories and their corresponding ID numbers for the search engine instance you are managing.
`list categories for instance ID`	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.
`list external search categories`	Lists the external search categories and their corresponding ID numbers for the search engine instance you are managing.
`list external search categories for instance ID`	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.
`list instances`	Lists the search engine instances and their corresponding ID numbers
`list objects`	Lists a summary of the searchable objects associated with the search engine instance you are managing.
`list objects for category ID`	Lists a summary of the searchable objects associated with the search category you specify. Specify the ID number corresponding to the desired search category.
`list objects for instance ID`	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.
`list objects for schedule ID`	Lists a summary of the searchable objects associated with the index schedule you specify. Specify the ID number corresponding to the desired index schedule.
`list params for instance`	Lists the parameters that are available for the `set param` command for the engine instance you are managing.
`list params for instance ID`	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.
`list schedules`	Lists the index schedules associated with the search engine instance you are managing.
`list schedules for instance ID`	Lists the index schedules associated with the search engine instance you specify. Specify the ID number corresponding to the desired search engine instance.
`list unassigned objects`	Lists the unassigned searchable objects (not associated with a search engine instance) and their corresponding ID numbers.
`manage instance`	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.
`manage instance ID`	Sets the context to the search engine instance you specify. Specify the ID number corresponding to the search engine instance you want to manage.

Table B–1 (Cont.) ECSF Command Line Administration Utility Commands

Command	Description
`register idplugin`	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.
`register idplugin for instance` *`ID`*	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.
`register object`	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.
`register object set "`*`fieldname`*`="`*`value`*`","`*`fieldname`*`="`*`value`*`"...`	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. 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, `"field name with \"escaped\" quotes"`.
`register unassigned object`	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.
`register unassigned object set "`*`fieldname`*`="`*`value`*`","`*`fieldname`*`="`*`value`*`"...`	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. 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, `"field name with \"escaped\" quotes"`.
`remove object from category` *`ID`*	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. 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.
`remove object` *`ID`* `from category` *`ID`*	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.
`remove object from instance`	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. In order to disassociate a searchable object from a search engine instance, both the object and the instance must be undeployed.

Table B–1 (Cont.) ECSF Command Line Administration Utility Commands

Command	Description
`remove object from instance ID`	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. In order to disassociate a searchable object from a search engine instance, both the object and the instance must be undeployed.
`remove object ID from instance`	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. In order to disassociate a searchable object from a search engine instance, both the object and the instance must be undeployed.
`remove object ID from instance ID`	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. In order to disassociate a searchable object from a search engine instance, both the object and the instance must be undeployed.
`remove object from schedule ID`	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. In order 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.
`remove object ID from schedule ID`	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. In order 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.
`set param "paramname"="value"`	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 "Managing Search with Oracle Enterprise Crawl and Search Framework" chapter in the *Oracle Fusion Applications 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, `"value with \"escaped\" quotes"`. You must issue the command while managing the search engine instance whose password parameters you want to set.

Table B–1 (Cont.) ECSF Command Line Administration Utility Commands

Command	Description
`set password param`	Sets password parameter values for the search engine instance. Pass in one password parameter and its password. See the "Managing Search with Oracle Enterprise Crawl and Search Framework" chapter in the *Oracle Fusion Applications 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, `"value with \"escaped\" quotes"`. You must issue the command while managing the search engine instance whose password parameters you want to set.
`showdetails for category` *`ID`*	Lists detailed information about the specified search category and the searchable objects associated with it.
`showdetails for unassigned object` *`ID`*	Lists the detailed information about the specified unassigned searchable object.
`showdetails`	Lists the detailed information for the search engine instance being managed. You must issue the command while managing a search engine instance.
`showdetails for instance` *`ID`*	Lists the detailed information about the specified search engine instance.
`showdetails for object` *`ID`*	Lists the detailed information about the specified searchable object.
`showdetails for param` *`ID`*	Lists the detailed information about the specified search engine instance parameter.
`showdetails for schedule` *`ID`*	Lists detailed information about the specified index schedule and the searchable objects associated with it.
`start schedule` *`ID`*	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. 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.
`stop schedule` *`ID`*	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.
`undeploy category` *`ID`*	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.
`undeploy object` *`ID`*	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.
`undeploy schedule` *`ID`*	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.
`unmanage`	Resets or exits the search engine instance context.
`unmanage instance`	Resets or exits the search engine instance context.

Table B–1 (Cont.) ECSF Command Line Administration Utility Commands

Command	Description
`update category ID`	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.
`update category ID set "fieldname"="value","fieldname"="value"...`	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, `"field name with \"escaped\" quotes"`. 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.
`update external category ID`	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.
`update external category ID set "fieldname"="value"`	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, `"field name with \"escaped\" quotes"`. 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.
`update instance`	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.
`update instance ID`	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.
`update instance set "fieldname"="value","fieldname"="value"...`	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, `"field name with \"escaped\" quotes"`.

Table B–1 (Cont.) ECSF Command Line Administration Utility Commands

Command	Description
`update instance` *`ID`* `set "`*`fieldname`*`="`*`value`*`","`*`fieldname`*`="`*`value`*`"...`	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, `"field name with \"escaped\" quotes"`.
`update object` *`ID`*	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.
`update object` *`ID`* `set "`*`fieldname`*`="`*`value`*`","`*`fieldname`*`="`*`value`*`"...`	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, `"field name with \"escaped\" quotes"`. You must issue the command while managing the search engine instance with which the searchable object is associated.
`update schedule` *`ID`*	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.
`update schedule` *`ID`* `set "`*`fieldname`*`="`*`value`*`","`*`fieldname`*`="`*`value`*`"...`	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, `"field name with \"escaped\" quotes"`. You must issue the command while managing the search engine instance with which the index schedule is associated.